UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITECNICA SUPERIOR

INGENIERIA TECNICA DE TELECOMUNICACION
TELEMATICA

PROYECTO FIN DE CARRERA

DEPARTAMENTO DE TELEMATICA

ESTUDIO EXPERIMENTAL DEL FUNCIONAMIENTO DE OSPF-MANET Y OLSR EN REDES MALLADAS MULTI-SALTO INALAMBRICAS

Autor: ANTONIO GUERRERO ESPARTERO

Director: CARLOS JESUS BERNARDOS CANO

Tutor: ANTONIO DE LA OLIVA DELGADO

Julio de 2010

Resumen

Hasta hace pocos años, la mayoría de las comunicaciones a través de red estaban basadas en cable, una pauta que está cambiando actualmente debido, principalmente, a la madurez alcanzada por las comunicaciones inalámbricas, que ahora ofrecen conexiones veloces, seguras, eficientes y fiables. Estas se encuentran en su momento de máxima expansión, claramente impulsadas por la aparición de los teléfonos móviles, PDAs, ordenadores portátiles, y un amplio número de dispositivos, todos ellos con soporte WiFi.
Dicha madurez convierte a las redes malladas inalámbricas en una opción muy a tener en cuenta a la hora de crear infraestructuras de red, que unida a su sencillez de instalación y su bajo coste, garantizan el éxito de esta solución.

La estructura mallada de este tipo de redes provoca que existan múltiples rutas a la hora de encaminar los datos a través de la red, así como un número variable de saltos para cada una de ellas; por este motivo, se hace necesaria la intervención de un protocolo de encaminamiento que se encargue de la distribución de los datos a través de la red de un modo eficiente y fiable.

Este proyecto fin de carrera se centra en el estudio teórico y práctico de dos de los protocolos diseñados para trabajar en este tipo de redes: OLSR, desarrollado específicamente para las redes malladas inalámbricas y OSPF-MANET, una modificación del protocolo OSPF básico, destinado para su uso en MANETs, pero también válido en el tipo de red tratada en este estudio. Para ello, se realizará un análisis del estado del arte relacionado con las redes malladas, así como de los protocolos mencionados.

Otra parte del estudio consistirá en una serie de experimentos, mediante el despliegue de una plataforma de pruebas, que permitan medir diferentes parámetros de red, como cargas de señalización y retardos introducidos por ambos protocolos en diferentes situaciones, con el fin de determinar su rendimiento y realizar una comparativa para decidir la solución más apropiada. Para ello se crearán varios escenarios de red que simulen lo más verazmente posible un caso real.

Conocida la limitación de direccionamiento existente en las redes IPv4 y teniendo en cuenta que el uso de dispositivos inalámbricos se extenderá aún más en el futuro, los experimentos también contemplarán el comportamiento de estos protocolos con IPv6, que sustituirá, en un futuro no muy lejano, a IPv4.
Todo esto permitirá conocer, más en profundidad, las peculiaridades de este novedoso tipo de redes, así como los distintos modos de encaminar tráfico a través de la red mediante los protocolos objeto del estudio, el comportamiento de éstos con IPv6, y las particularidades de contar con dispositivos limitados y su configuración.

Palabras clave: redes malladas, wireless, OSPF-MANET, OLSR, IPv6

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

ACRONIMOS

MANET	Mobile Ad-hoc Network
OSPF	Open Shortest Path First
OLSR	Optimized Link State Routing
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
WIFI	Wireless Fidelity
LAN	Local Area Network
PoE	Power Over Ethernet
ITS	Inteligent Transport System
IETF	Internet Engineering Task Force
PDA	Personal Digital Assistant
IP-sec	Internet Protocol security
QoS	Quality of Service
LS	Link State
SPF	Shortest Path First
IGP	Internet Gateway Protocol
AS	Autonomous System
RIP	Routing Information Protocol
LSA	Link State Advertisement
DD	Database Description Packets
DR	Designate Router
Ack	Acknowledgement
NBMA	Non-Broadcast Multiple Access
BDR	Backup Designated Router
CDS	Connected Dominating Set
BCDS	BiConnected Dominating Set
MDR	MANET Designate Router
BMDR	Backup MANET Designate Router
IANA	Internet Assigned Numbers Authority
MPR	MultiPoint Relay
TC	Topology Control
MID	Multiple interface declaration
RFC	Request for Comments
AODV	Ad hoc On Demand Distance Vector routing
DSR	Dynamic Source Routing
SDK	Software Development Kit
DHCP	Dynamic Host Configuration Protocol
LTS	Long Time Support
MAC	Media Access Control
CPU	Central Processing Unit
RAM	Random Access Memory
UDP	User Datagram Protocol
TCP	Transmission Control Protocol

PARTE I
INTRODUCCION

Capítulo 1 Introducción

1.1 Introducción

En la actualidad, existe una amplia variedad de protocolos de encaminamiento disponibles en el ámbito de las redes malladas, como son AODV [AODV], BATMAN [BATMAN], DSR [DSR], OSPF y OLSR, cuya importancia radica en la automatización del proceso de establecimiento de rutas en la red, evitando su creación manual.

Como ninguno de estos protocolos prevalece con claridad sobre los demás, el presente documento se centrará en el análisis, en profundidad, de las redes malladas inalámbricas y, más especialmente, en las ventajas e inconvenientes de dos de estos protocolos de encaminamiento, OSPF-MANET [OSPFM] y OLSR [OLSR], encargados de calcular y gestionar las rutas, garantizando la conectividad de los diferentes nodos pertenecientes a una red. Todo ello, con el fin de determinar, de una manera tanto cuantitativa como cualitativa, cual de ellos es el más apropiado para su uso en este tipo de redes.

Desde el punto de vista cualitativo se tendrán en cuenta aspectos como:
- Estabilidad
- Soporte
- Consumo de recursos
- Facilidad de instalación y configuración

Mientras que desde el punto de vista cuantitativo, se medirán parámetros de red como cargas de señalización y retardos, en diferentes escenarios y situaciones que simulen, lo más fielmente posible, un caso real.

La figura 1.1[1] muestra un ejemplo de red mallada inalámbrica, donde el nodo origen desea enviar un paquete a un nodo destino fuera de su radio de cobertura, por lo que tendrá que ser retransmitido a través de los nodos intermedios hasta alcanzar su destino.

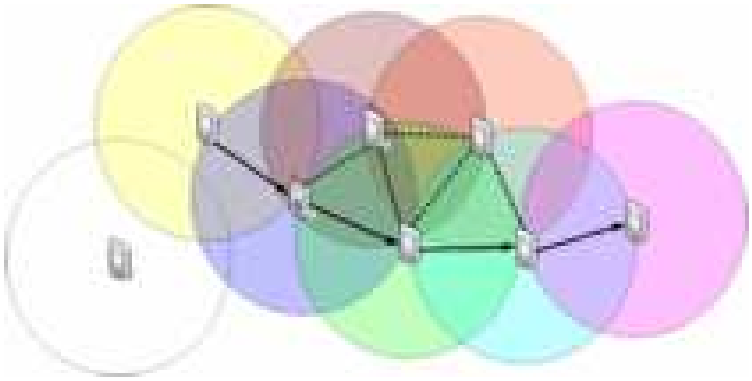


Figura 1.1 Ejemplo de red mallada inalámbrica

[1] Fuente de la imagen: http://www.igd.fhg.de/idb/en/projects/mobsec/manet/manet_small.png

1.2 Objetivos

Los objetivos principales de este proyecto pueden resumirse de la siguiente forma:

- Estudio del estado del arte de las diferentes tecnologías que serán empleadas en la realización del proyecto.

- Análisis de los dos protocolos de encaminamiento para redes malladas seleccionados, OSPF-MANET y OLSR.

- Adquisición de conocimientos de instalación y configuración en los dispositivos Linksys WRT54GL, con sus limitaciones de capacidad, utilizados en los escenarios de pruebas y migración del software necesario en estos dispositivos.

- Despliegue de una serie de escenarios, en los que probar y analizar el funcionamiento de ambos protocolos.

- Definición y realización de pruebas básicas de funcionamiento de ambos protocolos.

- Descripción, realización y obtención de medidas en los escenarios de prueba para su posterior análisis y comparación.

- Análisis y evaluación de las prestaciones de ambos protocolos de encaminamiento y la posterior presentación de los resultados obtenidos.

1.3 Fases del Desarrollo

La realización del proyecto se ha llevado a cabo en varias fases, que se describen brevemente a continuación:

a. Análisis teórico de OSPF-MANET y OLSR, con el fin de conocer en profundidad sus principales características y su funcionamiento.

b. Estudio del software necesario durante las pruebas, tanto en cuanto a la implementación de los protocolos, como al firmware utilizado en los routers, así como las aplicaciones que se utilizarán.

c. Realización de una prueba de funcionamiento básico de las implementaciones y herramientas en dos PCs.

d. Compilación del software necesario para su ejecución en los routers, tanto del firmware a utilizar en los routers, como de la implementación de OSPF-MANET.

e. Instalación de dicho software en los routers, y comprobación de su correcto funcionamiento.

f. Creación de la batería de pruebas y comprobación de su funcionalidad en un escenario preliminar con nueve nodos. Obtención de los primeros resultados.

g. Desarrollo de los escenarios de prueba definitivos y ejecución de la batería de pruebas en los mismos, para obtener los resultados definitivos de ambos protocolos.

h. Análisis y comparativa de los resultados obtenidos en las pruebas.

i. Análisis cualitativo y comportamiento comparado de ambos protocolos.

1.4 Medios Empleados

1.4.1 Hardware

Para la realización de este estudio se han empleado los siguientes medios hardware:

Dos PCs con sistema operativo Linux, empleados para comprobar el funcionamiento básico del software que se utilizará durante las pruebas, así como para la ejecución y configuración correspondiente a la batería de pruebas que se realizará en los diferentes escenarios.

Veinte routers Linksys WRT54GL versión 1.1 [WRT54], que emplean un procesador de 200 Mhz y utilizan el chip Broadcom 5352. Disponen de 4MB de memoria flash y 16MB de RAM, poseen el switch integrado en la CPU y están basados en Linux. En cuanto a conectividad, disponen de cinco puertos Ethernet, así como de una interfaz inalámbrica. La figura 1.2 muestra el aspecto del modelo en cuestión.

Figura 1.2 Router Linksys WRT54GL

1.4.2 Software

A continuación, se realizará una breve descripción acerca del software utilizado durante los experimentos. Este puede dividirse en dos partes, la primera destinada al software utilizado en los PCs y la segunda al destinado a los routers. Este contenido será ampliado en el capítulo 5, correspondiente al entorno de pruebas.

1.4.2.1 PCs

El sistema operativo seleccionado para los PCs fue Ubuntu, basado en Linux. Para la realización de las diferentes configuraciones y pruebas se hicieron necesarias distintas utilidades, todas ellas de código abierto, que se describen a continuación:

Herramientas de configuración: el comando *ip* permite configurar las interfaces de red de una manera muy completa.

Herramientas de gestión y conexión remota: los comandos *scp* y *ssh* ofrecen la posibilidad de acceder remotamente a otros equipos, lo que es imprescindible a la hora de realizar las pruebas, desde los PCs, en los routers. Por otro lado, el amplio número de dispositivos hardware utilizados durante las pruebas hace necesaria la utilización de scripts de configuración y ejecución para la automatización de las tareas. Para ello se desarrollaron programas escritos en *shell script*.

Herramientas de captura de tráfico: el programa *tshark* es un analizador de protocolos de red mediante línea de comandos, permite capturar paquetes de datos en tiempo real en una red o la lectura de paquetes desde una captura previamente salvada. Usa el formato de captura de *libpcap* o *tcpdump*. Esta utilidad servirá para la captura y el análisis de mensajes de señalización durante las pruebas realizadas mediante scripts. Por su parte, el programa *capinfos* permite la obtención de datos estadísticos a partir de ficheros de captura de paquetes previamente salvados; en este caso, se utilizarán, como entrada del programa, los ficheros generados por *tshark*.

1.4.2.2 Routers

El firmware seleccionado para los routers fue Openwrt [OPENWRT]. Para la realización de las pruebas y sus respectivas configuraciones se hizo uso de una serie de utilidades que se citan a continuación:

Herramientas de configuración: los comandos *ip* e *iwconfig* permiten configurar las interfaces de red, tanto cableadas como inalámbricas, de una manera muy completa. El comando *iptables* permite definir políticas de filtrado del tráfico que circula por la red, lo que permitirá simular diferentes topologías, descartando o aceptando paquetes provenientes de los nodos de la red. Con él se facilita enormemente la realización de las pruebas, en las que intervienen numerosos nodos equipados con interfaces inalámbricas, cuyo despliegue real para establecer los distintos escenarios sería inviable de otro modo, por motivos de espacio y manejo.

Implementación del protocolo OSPF-MANET: se utilizará para ello el software de enrutamiento Quagga [QUAGGA], que permite convertir cualquier máquina que disponga de un sistema operativo basado en Linux en un router. Por sí solo, este software no ofrece la funcionalidad requerida para este estudio, ya que sólo dispone de funcionamiento OSPF básico, así que será necesario aplicar el parche *OSPFv3 MANET MDR* (2 de Junio de 2008) [PARCHEMDR], para proporcionársela.

Implementación del protocolo OLSR: se utilizará el demonio olsrd [OLSRD], existente como paquete ya compilado para el entorno Openwrt.

1.5 Organización de la Memoria

La estructura de esta memoria se compone de cinco partes que se detallan a continuación:

Parte I: Introducción.
Introducción general, centrándose en los objetivos y motivaciones, además de presentar la propia organización de la memoria. Está formada por un único capítulo, el 1, con su mismo título.

Parte II: Estado del arte.
Descripción de las distintas tecnologías imprescindibles para la comprensión de este estudio, como son las redes malladas, el protocolo OSPF con su extensión MANET y, finalmente, el protocolo OLSR. Está formada por tres capítulos:

Capítulo 2. Redes Malladas Inalámbricas. Una descripción de las mismas y su funcionamiento.

Capítulo 3. Introducción del protocolo OSPF y análisis de las principales características de su extensión MANET.

Capítulo 4. Introducción y análisis de las principales características del protocolo OLSR.

Parte III: Descripción del trabajo realizado.
Presentación del trabajo realizado, formada por tres capítulos:

Capítulo 5. Entorno de pruebas. Utilidad del entorno, sus motivaciones y su arquitectura.

Capítulo 6. Escenarios y batería de pruebas. Descripción de los diferentes escenarios y las pruebas realizadas sobre los mismos.

Capítulo 7. Resultados de las pruebas. Se presentan los resultados tras la ejecución de la batería de pruebas sobre los escenarios, para OSPF-MANET y OLSR.

Parte IV: Conclusiones finales y trabajos futuros.
Se centrará en las conclusiones extraídas más importantes. También va acompañado de una propuesta sobre futuras líneas de investigación que se pueden seguir a partir de este proyecto. Está formada por un único capítulo, el 8, con su mismo título.

Parte V: Apéndices.
Contienen información detallada sobre compilación, configuración, instalación y scripts.

PARTE II
ESTADO DEL ARTE

Capítulo 2 Redes Malladas Inalámbricas

2.1 Introducción

En el ámbito de las redes inalámbricas, se pueden encontrar dos modos de operación diferentes, dependiendo de la utilidad a la que estén destinadas:

- Descentralizado, en el que cada nodo de la red puede comunicarse directamente con sus vecinos; éste es el modelo utilizado por las redes malladas inalámbricas.

- Centralizado, usado tradicionalmente en las redes inalámbricas locales, en el que la comunicación entre los diferentes nodos de la red es gestionada por un punto de acceso, que servirá de intermediario entre dichos nodos.

Aunque, por definición, las redes malladas inalámbricas son aquéllas que utilizan una topología mallada, en la práctica éstas están formadas por un conjunto de routers inalámbricos fijos, también conocidos como nodos mesh, que proporcionan una infraestructura distribuida a los nodos cliente sobre una topología parcialmente mallada [WMN07].

Debido a la utilización de este tipo de topología, es necesario utilizar un sistema de reenvío multisalto similar al empleado en las redes Ad-hoc inalámbricas.
Aunque este tipo de red es muy parecido al de las redes malladas inalámbricas, sus protocolos de encaminamiento no ofrecen un buen rendimiento al ser utilizados en el tipo de red que ocupa este estudio, motivado, principalmente, por la finalidad a la que ambas están destinadas; mientras que las redes malladas inalámbricas están diseñadas para un entorno estático, las redes Ad-hoc lo están para entornos multisalto de gran movilidad y cambios de topología dinámicos.

Hablar de las redes malladas inalámbricas es hablar de una tecnología emergente, que podría ofrecer la garantía de extender la conectividad a zonas de difícil acceso o, simplemente, evitar costosas obras de infraestructura de una manera sencilla. Estas pueden, de una manera efectiva, sin cables y con bajo coste, conectar ciudades mediante una tecnología, la inalámbrica, en la actualidad muy probada.

Tradicionalmente, las redes inalámbricas se basaban en un pequeño número de puntos de acceso, cableados o inalámbricos, que proporcionaban conectividad a los usuarios. En una red mallada inalámbrica, la conexión se extiende a decenas, o incluso cientos, de nodos inalámbricos, que intercambian datos entre sí, para garantizar la conectividad total en un área determinada.
Estos nodos poseen radiotransmisores que permiten enrutar e intercambiar datos mediante una interfaz inalámbrica, empleando los estándares WiFi comunes, conocidos como 802.11a [WIFI], 802.11b[WIFI], 802.11g[WIFI] y 802.11n[WIFI], con el fin de permitir la comunicación entre los usuarios, y, aún más importante, entre ellos mismos.
En la figura 2.1[2] se puede ver el ejemplo de un nodo mesh instalado en una farola.

[2] Fuente de la imagen: http://www.cisco.com/en/US/i/100001-200000/150001-160000/153001-154000/153655.jpg

Figura 2.1 Nodo mesh instalado en una farola

2.2 Protocolos de Encaminamiento Disponibles

Durante un proceso de comunicación a través de una red mallada inalámbrica, la información viaja desde el punto de origen al punto de destino, mediante saltos inalámbricos de un nodo mesh al siguiente. Estos nodos eligen automáticamente el camino, dependiendo del tipo de algoritmo de encaminamiento en que se base el protocolo utilizado, en un proceso conocido como encaminamiento dinámico.

2.2.1 Clasificación de los Protocolos de Encaminamiento

Podemos clasificar los diferentes protocolos en varias categorías, según sus propiedades:

- Centralizados o Distribuidos
- Estáticos o Adaptativos
- Reactivos o Proactivos

Una manera de categorizar los protocolos de encaminamiento consiste en dividirlos en algoritmos centralizados y distribuidos; en los algoritmos centralizados, todas las selecciones de ruta son realizadas en un nodo central, mientras que, en los distribuidos, el cálculo de las rutas es compartido entre los nodos de la red.

Otra posible clasificación consiste en tener en cuenta si las rutas cambian como respuesta a los patrones de entrada de tráfico. En los algoritmos estáticos, la ruta utilizada por los pares origen-destino es fija, independientemente de las condiciones del tráfico; sólo puede cambiar como respuesta al fallo de un nodo o de un enlace. Este tipo de algoritmo no consigue un buen rendimiento para muchos de los patrones de entrada de tráfico. La gran mayoría de redes de paquetes emplean alguna forma de enrutamiento adaptativo, mediante el que las rutas utilizadas para encaminar entre pares origen-destino pueden cambiar en respuesta a la congestión.

Una tercera clasificación, más relacionada con las redes inalámbricas, consiste en distinguir los algoritmos de encaminamiento entre proactivos y reactivos.
Los protocolos proactivos tratan de evaluar continuamente las rutas dentro de la red; así, cuando un paquete tiene que ser reenviado, la ruta ya es conocida y puede ser utilizada inmediatamente. La familia de protocolos *vector distancia* es un ejemplo de este caso.

Por su parte, los protocolos reactivos solicitan un procedimiento de cálculo de rutas únicamente bajo demanda, lo que significa que, cuando se necesita una ruta, se empleará algún tipo de procedimiento de búsqueda global. La clásica familia de algoritmos por inundación pertenece al grupo reactivo.
Los esquemas proactivos tienen la ventaja de que, cuando se necesita una ruta, el retardo introducido antes de que los paquetes actuales puedan ser enviados es muy pequeño. Por otro lado, tales esquemas necesitan cierto tiempo para alcanzar el régimen estacionario (steady state), lo que puede causar problemas si la topología cambia con frecuencia.

2.2.2 Propiedades Deseadas para los Protocolos de Encaminamiento

Puesto que los protocolos convencionales diseñados para las redes inalámbricas ofrecen un rendimiento muy pobre al utilizarse en redes malladas inalámbricas, es necesaria una opción nueva; la pregunta es: ¿qué propiedades debería poseer dicho protocolo?

Veamos a continuación respuestas a esa pregunta:

- Funcionamiento distribuido

El protocolo, por supuesto, debe ser distribuido; en ningún caso debe depender de un nodo de control centralizado, incluso en el caso de redes estacionarias o fijas. Esto garantiza que la caída de un nodo no provoque la caída total de la red, ya que, en la mayoría de los casos, existirá otro nodo alternativo a través del cual encaminar el tráfico.

- Libre de bucles

Con el fin de mejorar el rendimiento global, se busca un protocolo que garantice que las rutas suministradas estén libres de bucles. Esto evitará cualquier gasto innecesario de ancho de banda o consumo de CPU.

- Soporte de enlaces unidireccionales

Las comunicaciones mediante radiofrecuencia pueden provocar la formación de enlaces unidireccionales. El uso de estos enlaces, y no únicamente el de los bidireccionales, mejora el rendimiento del protocolo de encaminamiento.

- Seguridad

Las comunicaciones por radiofrecuencia son especialmente vulnerables a ataques de usurpación de identidad (impersonation attacks); así pues, para asegurar que el protocolo se comporta como es debido, se necesitan medidas de seguridad preventivas. La autenticación y la encriptación son probablemente las opciones idóneas, aunque surge un problema con la distribución de claves a través de los nodos en redes inalámbricas. En dicho campo, existe también debate sobre la utilización de IP-sec [IPSEC], que emplea túneles para transportar todos los paquetes.

- Ahorro de energía

Los nodos en una red inalámbrica son dispositivos de capacidad limitada, que pueden utilizar, incluso, la energía solar como fuente de energía; por ello, tienen una limitación muy grande en cuanto a alimentación, lo que hace muy recomendable el uso de algún tipo de modo de reposo (stand-by) para ahorrar energía. Por este motivo es importante que el protocolo de encaminamiento soporte este modo de funcionamiento o, al menos, incorpore una gestión eficiente del consumo de energía.

- Rutas múltiples

Con el fin de reducir la cantidad de reacciones producidas por los cambios en la topología de la red y la congestión, sería recomendable la utilización de rutas múltiples. Si una ruta pasa a ser inválida, es posible que alguna otra ruta almacenada pueda sustituirla y, de este modo, evitar que el protocolo inicie un nuevo procedimiento para descubrir rutas.

- Soporte para calidad de servicio (QoS)

Sería recomendable incorporar al protocolo algún tipo de soporte para ofrecer calidad de servicio, que podría ser empleado dependiendo del uso que se le vaya a dar a la red, por ejemplo, para ofrecer tráfico en tiempo real.

2.2.3 Protocolos disponibles

Como se mencionó en el capitulo 1, existen diferentes protocolos de encaminamiento disponibles, diseñados para ser utilizados en las redes malladas inalámbricas; aunque este estudio se centra en OSPF-MANET y OLSR, es necesario conocer las diferentes opciones existentes, por lo que, a continuación, realizaremos una breve descripción de los más conocidos.

2.2.3.1 AODV (Ad-hoc On Demand Distance Vector)

Este protocolo [AODV] pertenece al grupo reactivo, dado que las rutas hacia los destinos son generadas bajo demanda de los nodos origen, lo que implica que el conocimiento que se tiene de la red es local.
Unicamente mantiene las rutas hacia los destinos el tiempo requerido por los nodos origen, valiéndose de la utilización de números de secuencia para controlar la caducidad de las rutas.
AODV es capaz de encaminar tráfico unicast y multicast, siendo otra de sus principales características la de evitar la formación de bucles; además, se comporta muy bien frente a la escalabilidad de la red.

2.2.3.2 DSR (Dinamic Source Routing)

Algoritmo [DSR] basado en el concepto de encaminamiento en origen, en el que los paquetes incluyen en su cabecera un listado, establecido por el nodo origen, de los nodos que deben atravesar para alcanzar su destino; también perteneciente al grupo reactivo.
Cada nodo almacena en una cache los nodos destino y el listado de nodos a atravesar para alcanzarlos, que son actualizados en el momento en que se aprenden nuevas rutas.
El protocolo consta de dos mecanismos principales: descubrimiento de ruta y mantenimiento de ruta.
No utiliza mensajes periódicos, lo que ayuda a reducir la carga de señalización introducida en la red.
Una característica importante es que no presenta problemas al actuar con enlaces unidireccionales.

2.2.3.3 BATMAN (Better approach to mobile ad-hoc networking)

BATMAN [BATMAN] es un protocolo proactivo, diseñado para redes malladas Ad-hoc inalámbricas, lo que implica que mantiene continuamente la información acerca de todos los nodos existentes en la red, ya sean accesibles a través de uno o múltiples saltos.

La estrategia seguida por este protocolo consiste en determinar, para cada destino de la red, un nodo vecino a un solo salto de distancia, que pueda ser seleccionado como la mejor opción para comunicarse con el destino en cuestión.
Esto implica un coste computacional menor y una mayor eficiencia, ya que no se calcularán rutas completas, sino que cada nodo origen conocerá únicamente al nodo vecino, a un salto, más recomendable para alcanzar un destino, reduciendo, de este modo, el tamaño de la tabla de rutas, así como la carga de señalización.

La descripción de los protocolos OSPF-MANET y OLSR se omite en este capítulo, debido a que al formar la parte central del estudio, serán analizados en mayor profundidad en sus propios capítulos.

2.3 Ventajas de las redes malladas inalámbricas

Se describen a continuación las ventajas más notables de este tipo de redes:

- El uso de menos cableado significa un coste inferior para desplegar una red, particularmente en el caso de grandes áreas de cobertura.

- Cuantos más nodos formen parte de la red, más grande y rápida será, así como más redundante, lo que la hace más fiable ante caídas.

- Se basan en los mismos estándares WiFi (802.11a, b, g, n) que ya existen para la mayoría de las redes inalámbricas.

- Son la solución óptima en los casos en que las barreras arquitectónicas o la movilidad imposibilitan el cableado; parques, transporte público, etc.

- Un único nodo físicamente conectado a Internet es suficiente para ofrecer conectividad al resto de nodos de la red.

- Son útiles en lugares donde la cobertura inalámbrica pueda verse afectada o limitada, ya que el resto de nodos alrededor garantizará la conectividad a través de otro camino alternativo de los muchos disponibles.

- Son autoconfigurables, la red incorporará automáticamente los nuevos nodos a la estructura existente, sin necesidad de intervención de ningún administrador.

- Se recuperan automáticamente ante caídas; si un nodo que forma parte de un camino de enrutamiento se cae, automáticamente se generará un camino alternativo para subsanar el problema y garantizar la conectividad de la red.

- Las configuraciones malladas inalámbricas permiten una comunicación más eficiente en las redes locales, ya que los paquetes locales no tienen que pasar por un nodo central.

- Los nodos mallados inalámbricos son sencillos de instalar y desinstalar, haciendo la red muy adaptable y expandible, según mayor o menor cobertura sea necesaria.

2.4 Funcionamiento

La figura 2.2, a continuación, muestra el funcionamiento de una red mallada inalámbrica a la hora de compartir una conexión a Internet a lo largo de una LAN

(red local). Como se puede ver, únicamente es necesario un nodo con conexión a Internet para ofrecer conectividad al resto de nodos en la red. Este nodo cableado comparte la conexión a Internet inalámbricamente con el grupo de nodos más cercanos, los cuales, a su vez, la compartirán con sus grupos de nodos más cercanos, y así sucesivamente.

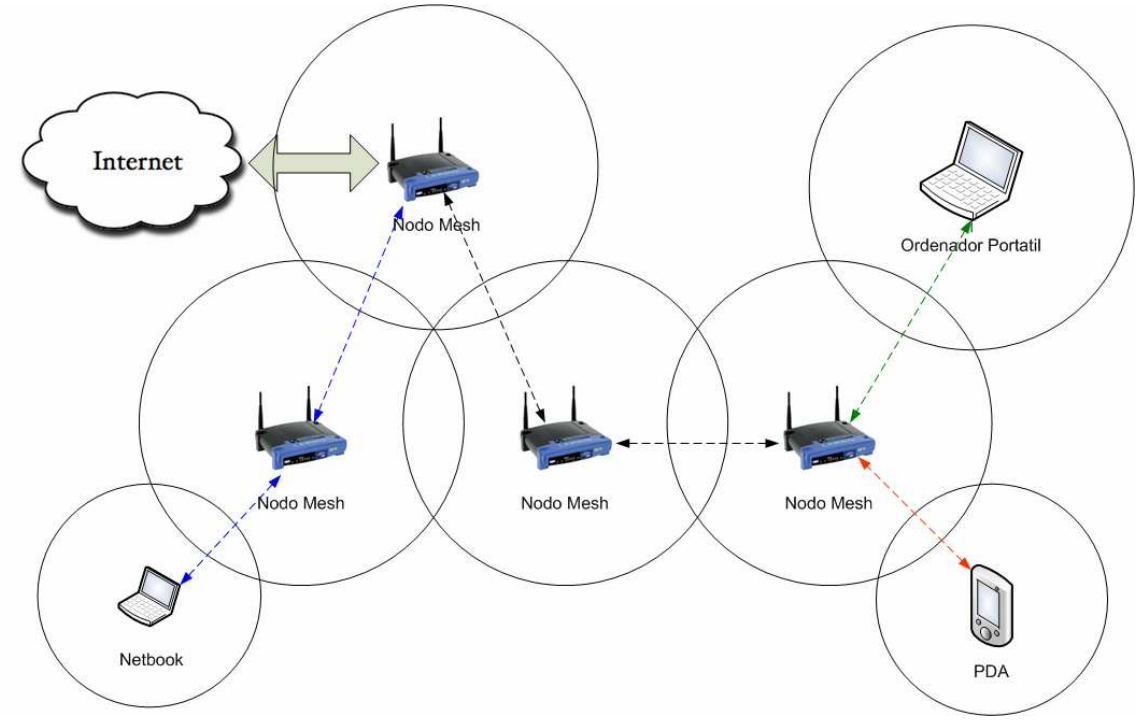


Figura 2.2 Esquema de funcionamiento de una red mallada inalámbrica con acceso a Internet

Esto significa que cada nodo, individualmente, no necesita estar conectado por cable. El único cableado necesario sería con una fuente de alimentación, bien tipo alimentadores tradicionales, bien baterías o paneles solares en exteriores. Los nodos a la intemperie están encapsulados en carcasas impermeables y resistentes, que pueden ser instaladas en cualquier lugar incluyendo farolas, tejados, etc.

Las redes malladas inalámbricas son eficaces a la hora de compartir una conexión a Internet debido a que, cuantos más nodos la formen, más cobertura será capaz de ofrecer, además de proporcionar una señal más fuerte y una comunicación más rápida.

Los nodos también pueden ofrecer conectividad a dispositivos cableados dentro de la red, como por ejemplo teléfonos IP, cámaras de video, PCs de sobremesa, etc. empleando el cableado Ethernet tradicional. La mayoría de los modelos de routers disponen de, al menos, dos puertos Ethernet; en este estudio hasta cinco, con el Linksys WRT54GL.
Además, mediante la tecnología PoE (Power Over Ethernet), los nodos pueden suministrar alimentación eléctrica a dispositivos autónomos como cámaras de vigilancia, etc., sin necesidad de usar una conexión eléctrica tradicional.

2.4.1 Nodos de Backhaul

Incluso en una red local mallada inalámbrica, llega el momento en que los datos tienen que viajar a través de un punto de acceso cableado para alcanzar Internet.

El envío de estos datos al punto de acceso cableado se conoce como Backhaul, troncal en español.
Las redes malladas inalámbricas pequeñas manejan el Backhaul sin necesidad de ninguna configuración especial; el problema surge en el caso de redes de mayor tamaño, como en ciudades o grandes compañías. En estos casos, algunos nodos, estratégicamente seleccionados, estarán destinados a comportarse como nodos de la troncal; el resto de nodos enviarán todos sus datos dirigidos a Internet a través de uno de estos nodos seleccionados, quienes reenviarán dicha información al punto de acceso cableado, evitando saltos innecesarios.

En el próximo apartado se verán algunas aplicaciones reales y potenciales de esta tecnología.

2.5 Aplicaciones

A continuación se describen los principales ámbitos de aplicación de las redes malladas inalámbricas:

- Municipios y ciudades

Gracias a las redes malladas inalámbricas, las ciudades pueden proporcionar acceso a los ciudadanos a un gran número de servicios públicos, mediante una conexión inalámbrica extensa de gran velocidad.
Un número cada vez mayor de núcleos urbanos ofrecen puntos de acceso WiFi públicos; las redes malladas inalámbricas permiten a las ciudades comunicar todos estos puntos de acceso, para cubrir la totalidad del municipio de un modo barato y simple.
La figura 2.3[3] muestra como una red mallada inalámbrica podría dar cobertura a una ciudad.

[3] Fuente de la imagen: http://www.strixsystems.com/images/illustrations/StrixWirelessIllustration.jpg

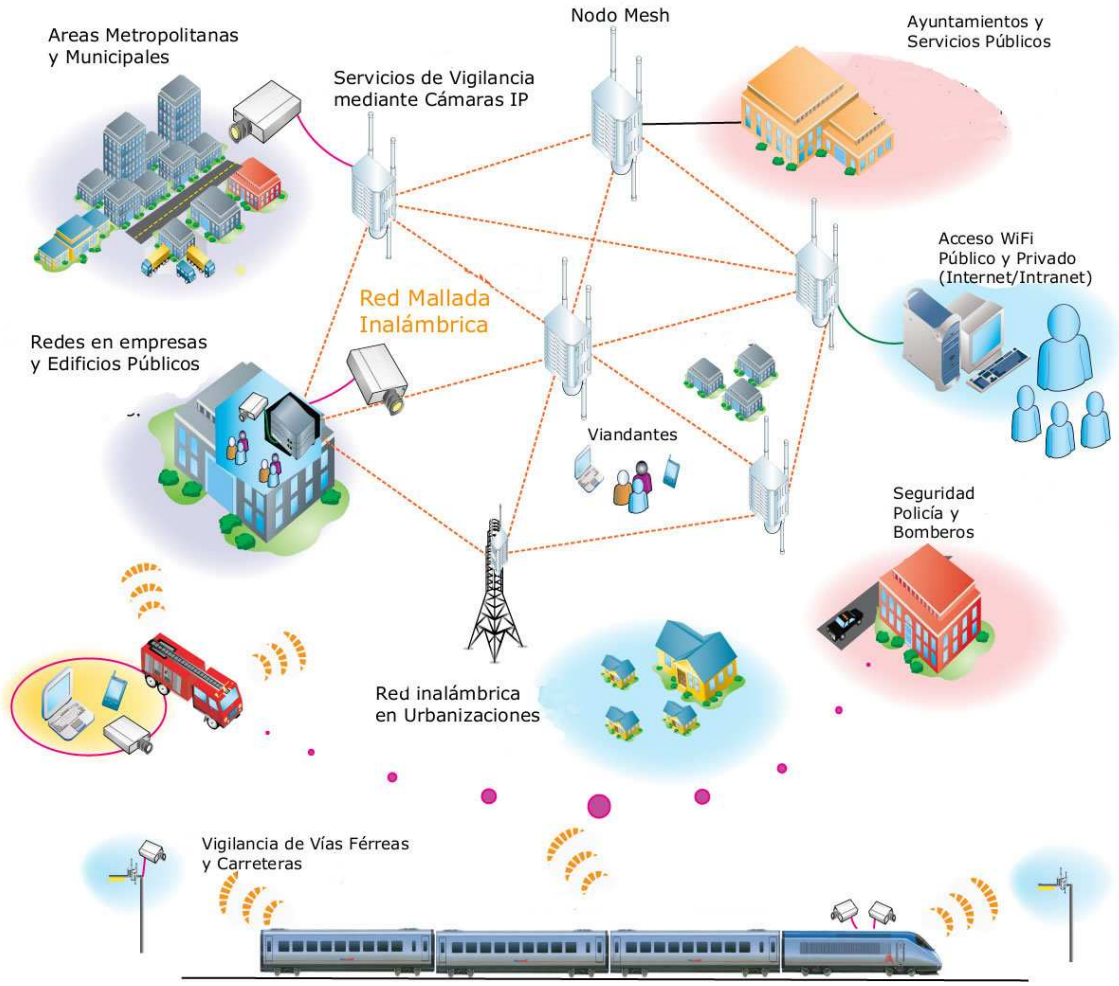


Figura 2.3 Aplicaciones de las redes malladas inalámbricas en ciudades

Algunas ventajas de las redes malladas inalámbricas municipales son:
Los viajeros y viandantes pueden acceder a su correo electrónico en el tren, en el parque, mientras cenan en un restaurante, etc.
Los funcionarios de obras públicas pueden consultar los diagnósticos de energía y suministro de agua de la ciudad, situando nodos inalámbricos en las instalaciones de tratamiento de aguas, alcantarillado y estaciones eléctricas, sin necesidad de costosas obras para abrir canalizaciones a través del suelo para instalar cables.
El personal de seguridad pública y emergencias puede acceder a redes virtuales seguras, dentro de la red general, para mantener la comunicación, incluso en situaciones en las que los servicios de telefonía fija y móvil estuviesen inoperativos. Mediante los nodos mallados inalámbricos instalados en farolas, semáforos, etc. la policía y los bomberos pueden mantenerse conectados a la red, incluso mientras se están desplazando.

- Países en desarrollo

Las redes malladas inalámbricas son muy útiles en países sin una infraestructura de cableado amplia, como el servicio telefónico o eléctrico. Nodos alimentados mediante energía solar pueden ser conectados a servicios de Internet por satélite o móvil, lo que mantendría a un pueblo entero conectado.

- Lugares aislados y zonas de difícil acceso

Hasta en países desarrollados podemos encontrar lugares aislados a los que no alcanza la cobertura de Internet. Las redes malladas inalámbricas podrían ofrecer

una solución a este problema, montando una serie de nodos desde el punto de conexión más cercano, hasta dar cobertura a la zona objetivo.

- Educación

Centros de estudios como las universidades pueden convertir sus campus en redes malladas inalámbricas, para ofrecer conectividad a sus estudiantes y profesores. Esta solución acaba con la necesidad de instalar cableado de red a través del campus o en edificios antiguos que no cuentan con preparación para ello. Con un pequeño número de nodos bien situados, cualquier miembro de la universidad puede tener acceso a la red. Un claro ejemplo podemos verlo en la Universidad Carlos III de Leganés. Los centros de estudios pueden, incluso, enlazar sus sistemas de seguridad a la red para vigilar sus cámaras de seguridad, o para mantener la comunicación del personal en situaciones de emergencia.

- Hostelería

Las conexiones a Internet de alta velocidad en hoteles y otros alojamientos son, en la actualidad, tan habituales que constituyen una norma y no una excepción. Las redes malladas inalámbricas son muy sencillas y rápidas de instalar, tanto en interiores como en exteriores, sin necesidad de remodelar la estructura, ni interrumpir la actividad del establecimiento.

- Emplazamientos temporales

Las zonas de obras pueden aprovechar la facilidad de instalación y desinstalación de las redes malladas inalámbricas. Los arquitectos e ingenieros pueden mantenerse conectados con la oficina, y las cámaras PoE garantizar la seguridad del recinto. Otras ventajas son la movilidad de los nodos y la facilidad de instalar otros nuevos según va avanzando el proyecto de construcción. Otros emplazamientos temporales tales como, ferias, congresos, conciertos, eventos políticos también pueden beneficiarse de la facilidad de puesta en marcha y desinstalación de este tipo de redes.

- Almacenes

En pocas palabras, no existe una forma efectiva de realizar un seguimiento de las existencias y la logística de envíos sin los escáneres portátiles con conexión Ethernet empleados en los almacenes modernos. Las redes malladas inalámbricas pueden asegurar la conectividad a lo largo y ancho de un almacén de grandes dimensiones con poco esfuerzo.

- Aplicaciones futuras

Los fabricantes de automóviles y las compañías de telecomunicaciones están trabajando en el desarrollo de un sistema de transporte inteligente (ITS), mediante redes malladas inalámbricas situadas a lo largo de calles y autopistas. Empleando una red automatizada de cámaras de vigilancia y sensores instalados en los vehículos, las autoridades de tráfico podrán supervisar el estado de las carreteras, el tráfico, los accidentes y otras situaciones de peligro.
Los fabricantes de chips y los desarrolladores de software ya venden soluciones de automatización para empresas y hogares que utilizan las redes malladas inalámbricas para sistemas de control y vigilancia remotos, control de temperatura, electrodomésticos y sistemas de entretenimiento.

El potencial de aplicación de este tipo de redes es ilimitado alcanzando cualquier campo imaginable.

Capítulo 3 OSPF-MANET

3.1 Introducción

Antes de tratar del protocolo OSPF-MANET [OSPFM], es necesario explicar el funcionamiento del protocolo OSPF original [OSPFv2], pues OSPF-MANET consiste en una serie de extensiones sobre el protocolo básico, con el fin de añadir soporte para su uso en redes inalámbricas; por ello, es imprescindible su comprensión previa para entender ciertos aspectos sin los cuales sería imposible.

OSPF es uno de los protocolos de *estado de enlace* [AEE] (Link State, LS) más conocidos y utilizados. Utiliza el algoritmo Shortest Path First (SPF) para calcular el camino más corto a cualquier destino conocido por el router que lo ejecuta.
Está clasificado como un protocolo de encaminamiento interno (Internet Gateway Protocol, IGP), lo que implica que distribuye la información de encaminamiento entre todos los routers dentro de un sistema autónomo (Autonomous System, AS), también llamado dominio de enrutamiento (routing domain).

Cada Router OSPF mantiene una base de datos, idéntica a la del resto de nodos, en la que se describe la topología completa del sistema autónomo al que pertenecen. A partir de esta base de datos, se obtienen las tablas de rutas mediante la creación del Shortest Path Tree [OSPFv2].

OSPF es capaz de recalcular rutas rápidamente al producirse cambios en la topología, introduciendo una mínima cantidad de tráfico de señalización en la red. Además, ofrece la posibilidad de reducir aún más ésta carga de señalización mediante la utilización de áreas (sección 3.7.2).

Al contrario que los protocolos basados en el algoritmo de *vector distancia* [AVD], como puede ser el caso de Routing Information Protocol [RIP] (más conocido como RIP), OSPF mantiene un mapa completo de la red. Este mapa se conoce como base de datos LS (LS database), que es una base de datos distribuida; cada router dentro del mismo sistema autónomo o área (si éstas son utilizadas) posee una base de datos idéntica. Para mantenerla actualizada, es necesario un esfuerzo que garantice su sincronización entre todos los routers.
De todos los tipos de paquetes definidos para OSPF, la mayoría están destinados a la sincronización de esta base de datos.

3.1.1 Definición de los Conceptos más Usados

Para la correcta comprensión de este capítulo, es necesario definir algunos conceptos clave, que serán utilizados durante la explicación:

- Sistema autónomo (AS)

Conjunto de routers que intercambian información de encaminamiento mediante un protocolo de encaminamiento común.

- Router ID

Identificador numérico de 32 bits asignado a cada router que ejecuta el protocolo OSPF. Este número únicamente identifica a un router dentro de un AS.

- Redes punto a punto

Red formada únicamente por una pareja de routers.

- Redes broadcast

Redes capaces de soportar varios (más de dos) routers asociados, además de poder enviar un único mensaje a todos los routers de la misma (broadcast).
En este tipo de redes, los vecinos son descubiertos dinámicamente mediante el protocolo Hello, valiéndose de esta capacidad de broadcast.
Cada par de routers en una red broadcast será capaz de comunicarse directamente.

- Redes No broadcast

Redes capaces de soportar varios (más de dos) routers asociados, pero sin capacidad broadcast. En estas redes, los vecinos son mantenidos mediante el protocolo Hello.
Debido a la falta de capacidad de broadcast, será necesaria cierta información de configuración extra para apoyar el descubrimiento de vecinos. En este tipo de redes, los paquetes OSPF, que son normalmente enviados mediante multicast, tendrán que ser enviados a cada vecino de uno en uno.
OSPF ofrece dos modos de funcionamiento sobre las redes no broadcast: el primero, conocido como *non-broadcast multi-access* (NBMA), simula el funcionamiento de OSPF en una red broadcast; el segundo, conocido como *punto a multipunto,* considera a la red no broadcast como un conjunto de enlaces punto a punto.
Las redes no broadcast son nombradas como redes NBMA o *punto a multipunto*, dependiendo del modo de operación OSPF ejecutado sobre la red.

- Routers vecinos

Se dice de dos routers que son vecinos cuando tienen interfaces asociadas a una misma red; las relaciones entre routers vecinos son descubiertas dinámicamente y mantenidas en OSPF mediante el protocolo Hello.

- Adyacencia

Relación establecida entre routers vecinos seleccionados, con el fin de intercambiar información de encaminamiento. No todas las parejas de routers vecinos tienen que ser adyacentes.

- Link State Advertisement (LSA)

Unidad de datos que describe el estado local de un router o una red. En el caso de un router, incluye el estado de sus interfaces y adyacencias. Cada LSA es inundado a través del dominio de routing.
El conjunto de los LSAs de todos los routers y redes forma la base de datos de estado de enlace definida por este protocolo.

- Protocolo Hello

Su función es establecer y mantener las relaciones entre routers vecinos. Su uso en redes broadcast permite el descubrimiento dinámico de vecinos.

- Inundación (flooding)

Consiste en un empleo masivo del broadcast, donde el router origen envía un paquete a sus vecinos (en el caso inalámbrico, significa a todos los nodos dentro del radio de transmisión); éstos lo retransmitirán a su vez a sus propios vecinos y así sucesivamente, hasta que el paquete haya sido recibido por todos los nodos de la red.

- Router designado (DR)

Cada red broadcast o NBMA que posee al menos dos routers asociados tendrá un router designado, el cual es elegido por el protocolo Hello. Su principal función, aunque tiene otras responsabilidades, será la de generar un LSA para la red.

Este concepto permite la reducción del número de adyacencias necesarias en este tipo de redes, lo que facilita la disminución del tráfico de encaminamiento y la reducción de la base de datos de estado de enlace.

3.2 Funcionamiento de OSPF

El protocolo OSPF enruta los paquetes IP basándose únicamente en la dirección IP de destino almacenada en la cabecera del paquete IP. Estos son enrutados tal cual, es decir, no vuelven a ser encapsulados con una nueva cabecera, mientras viajen a través del sistema autónomo.
OSPF es un protocolo de encaminamiento dinámico, detecta rápidamente cambios en la topología del dominio de enrutamiento (como pueden ser fallos en las interfaces de los routers) y calcula nuevas rutas libres de bucles tras un periodo de convergencia. Este periodo es corto y conlleva un tráfico de encaminamiento mínimo.

Como se mencionó anteriormente, en un protocolo de estado de enlace cada router mantiene una base de datos, conocida como base de datos LS, que describe la topología del dominio de enrutamiento; esta base de datos es idéntica en todos los routers y está formada por los estados locales (generalmente, las interfaces funcionales y los vecinos alcanzables) de cada uno de los routers.
Cada router distribuye, mediante inundación, su estado local a través del sistema autónomo.

A partir de la base de datos LS, cada router crea una estructura de árbol con los caminos más cortos, usándose a sí mismo como raíz; ésta estructura es conocida como Shortest Path Tree (SPT), generando la ruta a cada destino del sistema autónomo. La información recibida desde el exterior aparece como hojas en el árbol.

OSPF define un coste que permite diferenciar a la hora de elegir una ruta u otra, éste se describe mediante una métrica de una sola dimensión; cuando existen varias rutas al mismo destino con el mismo coste, el tráfico se distribuye por igual a través de éstas.

OSPF ofrece la capacidad de agrupar varias redes en un conjunto denominado área. El encaminamiento dentro de un área está determinado únicamente por la topología de la misma; ésta está oculta para el resto del sistema autónomo y no es compartida, lo que conlleva una reducción significativa en el tráfico de encaminamiento. Un área consiste en una generalización de subred IP.

3.3 La Base de Datos LS

Como se mencionó anteriormente, la base de datos LS representa el mapa de la topología de la red; dentro de un área, esta base de datos es sincronizada entre todos los routers.
Cuando un router se une por primera vez a un área, copia la base de datos LS de alguno de sus vecinos (resultando en que los routers consiguen adyacencia, como se explica en la sección 3.3.5).
Después de la sincronización inicial, cada router transmite mediante inundación (flooding) sus actualizaciones de rutas (routing updates) a los demás routers pertenecientes al área [OSPFDBEO]; de este modo, todos los routers podrán analizarlas y compararlas con su propia visión de la topología.

3.3.1 La Tabla de Rutas

La base de OSPF es el algoritmo SPF. Como su propio nombre indica, el protocolo Open Shortest Path First intenta calcular el camino más corto a cualquier destino posible. La fuente de estos cálculos es la visión de la red por parte de los routers (la base de datos LS).
Basándose en las entradas de la base de datos, el router crea una estructura SPF organizada en árbol (SPF tree) desde él mismo hasta el destino dado.
Apoyándose en este "SPF tree" el router decide a qué vecino enviará el paquete en cuestión. En otras palabras, la base de datos LS y el cálculo del "SPF tree" sirven para un único propósito: decidir a quien le será reenviado un paquete dado. El SPF tree ofrece una tabla de rutas al router.

Es posible asignar un coste a cada enlace de la red y tomar esas métricas en cuenta a la hora de calcular el SPF tree. Este método tiene la ventaja de que pueden tenerse en cuenta factores como ancho de banda o retrasos.

El cálculo de los caminos más cortos (SPF) es tarea del algoritmo *Dijkstra*, aplicándolo sobre la información mantenida en la base de datos.

3.3.2 Algoritmo Dijkstra

El algoritmo Dijkstra [DIJKS] busca encontrar los caminos más cortos desde un nodo origen dado al resto de nodos de la red, desarrollando los caminos en orden de longitud creciente.
El algoritmo sigue una serie de etapas, donde en la etapa *K*-esima, los caminos más cortos a los *K* nodos más cercanos (el menor coste en distancia) al nodo origen han sido determinados. Dichos nodos se agrupan en el conjunto *T*.
En la etapa (*K* + 1), el nodo que no pertenezca a *T* con el camino más corto desde el nodo origen es añadido a este grupo.
A medida que cada nodo es añadido a *T*, su camino desde el origen queda definido.

El algoritmo puede ser descrito formalmente como sigue:

- Definiciones

N = conjunto de nodos pertenecientes a la red
s = nodo origen
T = conjunto de nodos incorporados, hasta el momento, por el algoritmo
w(*i,j*) = coste del enlace desde el nodo i al nodo j; *w*(*i,j*) = infinito, si los nodos no están directamente conectados; *w*(*i,j*) >= 0, si están directamente conectados.
L(*n*) = coste del camino con menor coste desde el nodo *s* al nodo *n*, conocido en el algoritmo como "terminación".

El algoritmo tiene tres etapas; la segunda y la tercera se repiten hasta que *T* = *N*, es decir, hasta que los caminos finales han sido asignados a todos los nodos de la red.

- Etapa 1 Inicialización

T = {*s*}
El conjunto de nodos incorporados hasta el momento sólo incluye al origen.
L(*n*) = *w*(*s*,*n*), para *n* distinto de *s.*
Los costes del camino inicial a los nodos vecinos son simplemente los costes del enlace.

- Etapa 2 Obtención del siguiente nodo

Encontrar el nodo vecino que no esté en *T* con el camino de menor coste desde el nodo *s* e incorporarlo a *T*. También se incorporará el enlace desde ese nodo hasta un nodo de *T* que forma parte del camino. Esto puede ser expresado como:
Encontrar *x* no perteneciente a *T* tal que $L(x)$ = mínimo $L(j)$, donde *j* no pertenece a *T*.

- Etapa 3 Actualización de los caminos de menor coste

$L(n) = \min[L(n), L(x) + w(x,n)]$, para todo *n* no perteneciente a *T*.

El algoritmo termina cuando todos los nodos han sido añadidos a *T*.

En la figura 3.1[4] puede verse un ejemplo del proceso de cálculo seguido por el algoritmo Dijkstra, a partir de un grafo de nueve nodos.

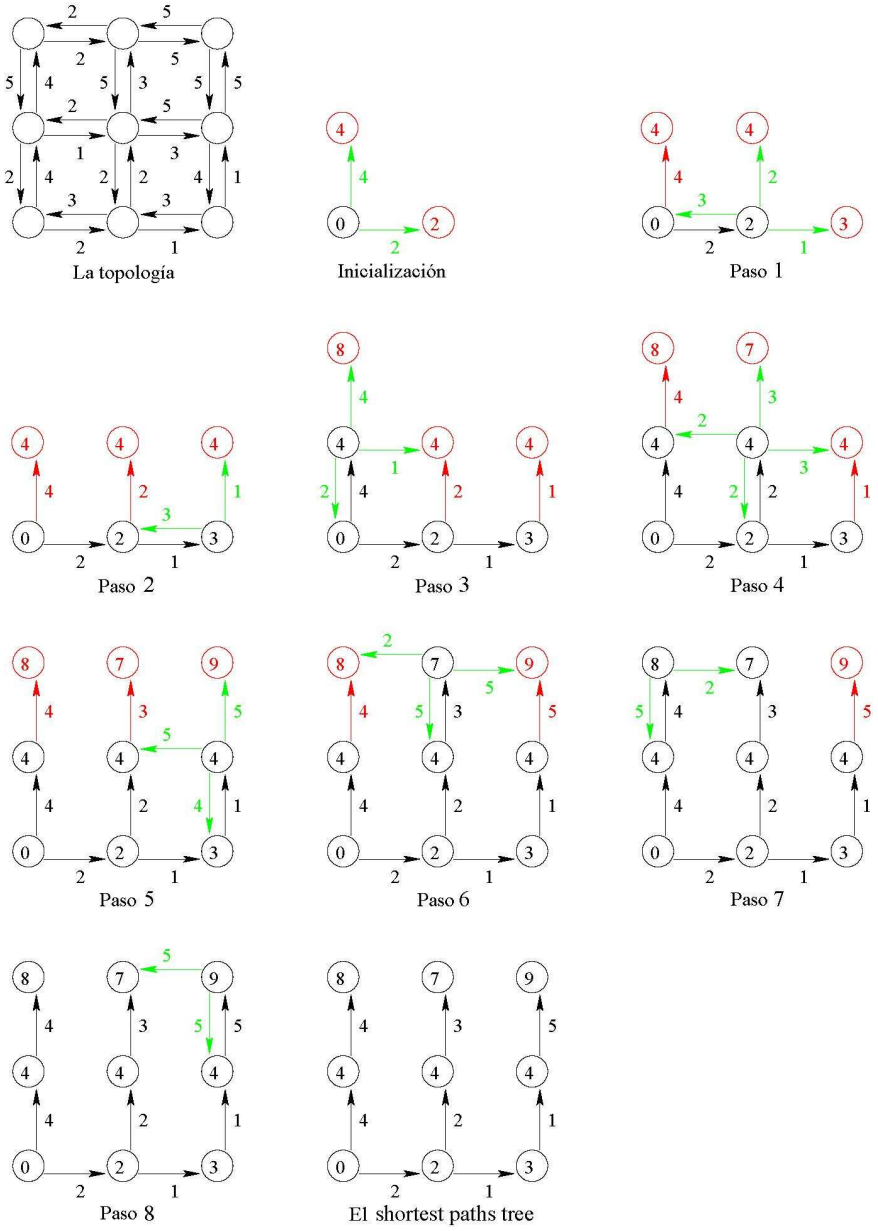


Figura 3.1 Ejemplo del proceso de cálculo del algoritmo Dijkstra

[4] http://www.cacr.caltech.edu/~sean/projects/stlib/html/shortest_paths/DijkstraFigure.jpg

3.3.3 El Link State Advertisement (LSA)

Por definición, los LSAs son los mensajes de actualización de rutas. Estos mensajes describen la conectividad de los routers que ejecutan OSPF. Existen varios tipos de mensajes LSA, un router genera estos mensajes para indicar sus adyacencias, prefijos asociados con las redes conectadas, etc. Los LSAs son transportados en los paquetes de actualización LS (LS Update packets).
Los LSAs son enviados mediante inundación a través de los enlaces en los que el router receptor es adyacente (descrito en la sección 3.3.5) al router transmisor. Tan pronto como dos routers comienzan la sincronización inicial de base de datos, los futuros LSAs serán enviados por inundación a través de este enlace, mientras se mantenga la adyacencia que se inició.

Para ilustrar el concepto de los LSAs, se va a describir uno de estos tipos de mensaje, el router LSA. Este tipo de LSA describe al área el estado de las interfaces del router, mediante la inclusión de descripciones sobre los vecinos completamente adyacentes (fully adjacent) en la interfaz.
Cada interfaz generará este tipo de LSAs, y se generarán nuevas instancias del LSA, por ejemplo, cuando se formen nuevas adyacencias. Un vecino adyacente está determinado por los descriptores de enlace (link descriptors), indicando, entre otros, el coste de alcanzar al vecino, además del router ID de este último.

3.3.4 Sincronización Inicial de la Base de Datos LS

Para unirse a una nueva red, un router debe aprender la base de datos de dicha red; esto se consigue mediante la realización de una sincronización con la base de datos de uno de sus nuevos vecinos.
Mencionar que, dependiendo del tipo de interfaz utilizada, el vecino con el que realizará dicha sincronización no será aleatorio. Este aspecto será discutido posteriormente.

Los routers, al sincronizarse, intercambian paquetes de descripción de base de datos (database description packets, DD), describiendo sus LSAs actuales para el área. Si un router descubre alguna discrepancia, solicitará ese LSA problemático a su vecino.
El proceso de intercambio de base de datos (database exchange) deja a ambos routers con una base de datos LS idéntica, pasando a considerarse completamente adyacentes (fully adjacent). A este proceso se le conoce como sincronización inicial.

3.3.4.1 Mantenimiento de la Base de Datos

La función de sincronización inicial de la base de datos consiste en asegurar que los nuevos routers que se unan posean una visión actualizada de la topología de la red. Este proceso se realiza únicamente una vez por cada nuevo router.
Para adaptarse a los cambios producidos después de la sincronización inicial, los routers intercambiarán LSAs con regularidad.

3.3.5 Adyacencias

Se dice que dos routers son adyacentes cuando forman una relación con el propósito de intercambiar información de encaminamiento. Pasan a ser adyacentes en el momento en que el proceso de sincronización de la base de datos es iniciado, puesto que a partir de ese momento los LSAs serán enviados usando inundación a lo largo del enlace.

Se dice que son completamente adyacentes (fully adjacent) cuando sus bases de datos están sincronizadas. Únicamente los vecinos completamente adyacentes son incluidos en los mensajes LSA. En redes en las que se utiliza el esquema de "router designado" (Designate Router, DR), los routers sólo formarán adyacencias con el DR; de este modo, en este tipo de redes el número de adyacencias es mínimo.
El número de adyacencias formadas es, habitualmente, un parámetro clave cuando medimos el impacto de la carga producida por OSPF sobre diferentes redes.

El número de vecinos adyacentes es proporcional a la cantidad de tráfico de encaminamiento que será necesario enviar a través de la red. La cantidad de routers adyacentes supone una desventaja en el modelo de interfaz punto a multipunto que se describe en la sección 3.5.4.

3.3.5.1 El Protocolo HELLO

Un router anuncia su existencia al resto de routers mediante el envío de paquetes hello, siguiendo un intervalo regular de tiempo. Cuando un nuevo router se une a la red, emite un paquete hello indicando su presencia; en ese momento, no sabrá de la existencia de ningún vecino y, por ello, la lista de vecinos que incluirá en este paquete estará vacía.
Los routers vecinos recibirán el paquete y descubrirán al nuevo router; cuando el nuevo router recibe un paquete de un vecino que escuchó su hello, el nuevo router ID estará incluido en el mismo. De este modo, el nuevo router sabe que ese vecino ha escuchado su mensaje hello y crea una entrada para éste en su estructura de datos de vecinos locales (local neighbor data structure). Como el router ahora sabe que se pueden escuchar entre ellos, éste marca la comunicación como bidireccional.

A partir de entonces, en los paquetes hello siguientes originados por el nuevo router será incluido el Router ID del vecino.
Esta es la manera en que el protocolo establece las relaciones entre vecinos.

3.3.5.2 Estados de los Vecinos

Cuando dos routers pasan a ser vecinos, progresan a través de diferentes estados que describen su relación recíproca.

Un tipo de estos estados es conocido como "TWOWAY" cuando se ha establecido conectividad bidireccional.

Si los routers deciden pasar a ser adyacentes, entran en el estado "EXCHANGE", que se produce cuando la sincronización inicial de la base de datos comienza.

Otro estado, conocido como "FULL", se produce cuando ambos routers son adyacentes y están completamente sincronizados (fully synchronized).

Por ultimo, el estado "DOWN" indica que los routers no se conocen, o dejan de hacerlo.

Estado	**Situación**
TWOWAY	Existe conectividad bidireccional
EXCHANGE	Iniciada sincronización inicial
FULL	Existe adyacencia
DOWN	No existe conectividad

Tabla 3.1 Resumen de estados entre vecinos en OSPF

3.4 Paquetes OSPF

Los paquetes OSPF se envían, a través de la red, encapsulados en paquetes TCP/IP y usan el puerto 89 asignado por IANA [IANA] (Internet Assigned Numbers Authority).

3.4.1 Tipos de Paquetes

La tabla 3.2 muestra un resumen de los diferentes tipos de paquetes definidos para OSPF. Simplificando, estos paquetes pueden ser divididos en dos categorías: el paquete Hello, usado para establecer y mantener las relaciones con los vecinos, y el resto de paquetes, cuyo objeto es asegurar que los routers mantengan una visión consistente de la red.
Por lo tanto, de los cinco tipos de paquetes definidos, cuatro son utilizados para mantener la base de datos LS (link state database).

Los paquetes OSPF son enviados únicamente a través de adyacencias (con la excepción de los paquetes Hello, que son utilizados para el descubrimiento de las mismas). Esto significa que todos los paquetes de encaminamiento viajan un único salto IP.

Nombre del paquete	Tipo	Función
Hello	1	Descubrir y mantener vecinos
Database Description	2	Resumir el contenido de la base de datos
Link State Request	3	Descarga de la base de datos
Link State Update	4	Actualización de la base de datos
Link State Ack	5	Confirmación de inundación

Tabla 3.2 Resumen de los diferentes tipos de paquetes usados en OSPF

Todos los paquetes de encaminamiento comienzan con una cabecera estándar; a continuación se describirán los campos de datos incluidos en la misma, para después citar los campos específicos de cada tipo de paquete.

3.4.1.1 Cabecera OSPF Genérica

Cada paquete OSPF tiene una cabecera estándar de 24 bytes; ésta contiene toda la información necesaria para determinar si deberá ser aceptado en un proceso futuro. En la figura 3.2 puede verse el formato de la cabecera común de OSPF.

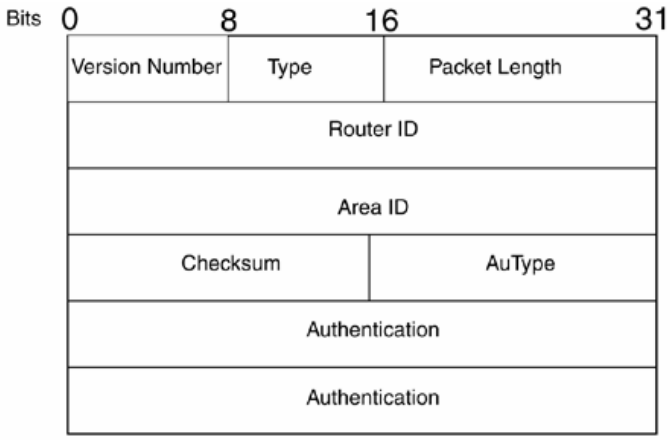


Figura 3.2 Cabecera OSPF Genérica

- Version
  El numero de versión de OSPF utilizado.

- Type
  El tipo de paquete OSPF (vistos en la tabla 3.2)

- Packet length
  El tamaño del paquete OSPF en bytes, incluyendo la cabecera OSPF estándar.

- Router ID
  El router ID del origen del paquete OSPF.

- Area ID
  Identificador de 32 bits del área a la que pertenece el paquete. Todos los paquetes OSPF están asociados a una única área, la mayoría solamente viajan un único salto.

- Checksum
  El checksum IP estándar del contenido total del paquete, empezando con la cabecera OSPF pero excluyendo el campo de autenticación de 64 bits.

- AuType
  Identifica el procedimiento de autenticación que será utilizado para el paquete.

- Authentication
  Campo de 64 bits usado por el esquema de autenticación.

3.4.1.2 Paquete Hello

Este paquete de tipo 1 es enviado periódicamente en todas las interfaces, con el fin de establecer y mantener las relaciones entre vecinos. Particularmente, estos paquetes son enviados mediante multicast, en redes con capacidad multicast o broadcast, habilitando el descubrimiento dinámico de routers vecinos.

Todos los routers pertenecientes a una misma red deben acordar ciertos parámetros (“Network mask”, “HelloInterval”, y “RouterDeadInterval”). Estos están incluidos en los paquetes Hello, por lo que diferencias en estos campos pueden impedir la formación de relaciones entre vecinos.

- Network mask
  La máscara de red asociada con la interfaz.

- Options
  La funcionalidad opcional soportada por el router.

- HelloInterval
  El número de segundos entre el envío de paquetes Hello para el propio router.

- Rtr Pri
  La prioridad de router, utilizada en la elección del Router Designado y de Backup. Si es 0, el router no podrá ser elegido como tal.

- RouterDeadInterval
  El número de segundos antes de declarar como caído (Down) a un router en silencio.

- Designated Router
  La identidad del Router Designado para esta red, desde el punto de vista del router emisor. El DR está identificado por la dirección IP de su interfaz de red. Si es 0.0.0.0 significa que no existe Router Designado.

- Neighbor
  El Router ID de cada nodo de la red desde el que se ha recibido un paquete Hello recientemente (en los últimos segundos establecidos por el campo "RouterDeadInterval").

La figura 3.3 muestra el proceso de descubrimiento de un nuevo vecino que se asocia a la red OSPF.

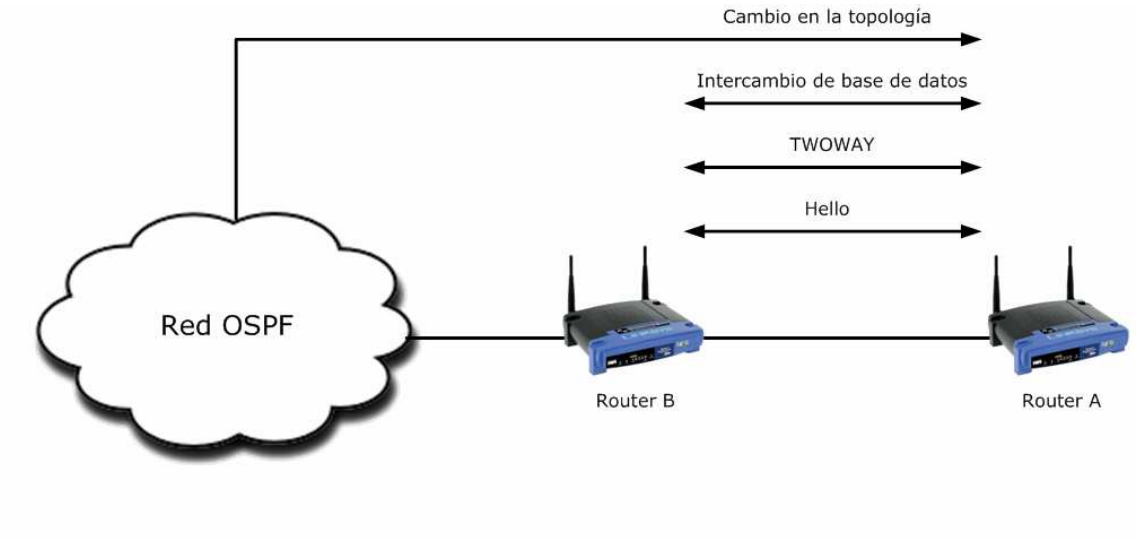


Figura 3.3 Descubrimiento de un vecino nuevo en OSPF

3.4.1.3 Paquete de Descripción de Base de Datos (Database Description)

Estos paquetes de tipo 2 son intercambiados en el momento en que se inicia una adyacencia. Describen el contenido de la base de datos LS, para lo que pueden utilizarse múltiples paquetes. Con este fin, se emplea un proceso de respuesta mediante sondeo. Uno de los routers es elegido como Maestro y el otro como Esclavo, el Maestro envía paquetes Database Description, que son confirmados por los paquetes Database Description enviados como respuesta por el Esclavo. Las respuestas están enlazadas a los paquetes origen mediante el identificador "DD secuence number".

El formato de los paquetes DD es muy parecido, tanto al de los paquetes Link State Request, como al de los paquetes Link State Acknowledgment. La parte principal de todos ellos es una lista de elementos, donde cada uno describe una parte de la base de datos LS.

- Interface MTU
  Tamaño en bytes del datagrama IP, sin fragmentar, más largo que puede ser enviado desde la interfaz asociada.

- Options
  La funcionalidad opcional soportada por el router.

- I-bit
  El bit de Inicio; cuando vale 1, indica que el paquete es el primero en la secuencia de paquetes DD.

- M-bit
  El bit More; cuando vale 1, indica que siguen más paquetes DD después de éste.

- MS-bit
  El bit Maestro/Esclavo; cuando vale 1, indica que el router es el Maestro durante el proceso de intercambio de base de datos.

- DD sequence number
  Permite secuenciar la colección de paquetes DD. El valor inicial (indicado por el bit de Inicio) debe ser único. A partir de éste, es incrementado hasta que la descripción de la base de datos ha sido totalmente enviada.

El resto del paquete consiste en una lista (puede que parcial) de las partes de la base de datos LS. Cada LSA en la base de datos esta descrito por su cabecera LSA.

3.4.1.4 Paquete Link State Request

Este paquete de tipo 3 es utilizado después del proceso de intercambio de paquetes DD entre routers vecinos; en ese momento, un router puede encontrar que ciertas partes de su base de datos LS están desactualizadas. El paquete Link State Request sirve para solicitar a un router vecino sus partes de la base de datos LS que se encuentran actualizadas (puede ser necesario el envío de varios paquetes).

Un router que envía un paquete Link State Request conoce la instancia precisa de la parte de la base de datos que esta solicitando. Cada instancia está definida por su "LS secuence number", "LS checksum" y su "LS age", aunque estos campos no están especificados en el paquete Link State Request.

Cada LSA solicitado está especificado por su "LS type", "Link State ID" y "Advertising Router", lo que únicamente identifica al LSA, pero no a su instancia. Los paquetes Link State Request son entendidos como peticiones para la instancia más reciente.

3.4.1.5 Paquete Link State Update

Este paquete de tipo 4 se encarga de la inundación de los LSAs. Cada paquete transporta una colección de LSAs a un salto de distancia de su origen.

Estos paquetes pueden ser enviados mediante multicast en redes que soporten multicast o broadcast. Para hacer el proceso de inundación fiable, los LSAs inundados son confirmados mediante paquetes Link State Acknowledgment. Si se hace necesaria la retransmisión de ciertos LSAs, éstos serán enviados directamente al vecino.

El cuerpo de un paquete Link State Update consiste en una lista de LSAs, donde cada LSA comienza con una cabecera común de 20 bytes.

3.4.1.6 Paquete Link State Acknowledgment

Este paquete de tipo 5 se encarga de la confirmación de los LSAs inundados, con el fin de hacer el proceso fiable.

El formato de este paquete es similar al paquete DD. El cuerpo de ambos paquetes es, simplemente, una lista de cabeceras LSA; cada LSA confirmado está descrito por dicha cabecera, y contiene toda la información necesaria para identificar tanto al LSA como a su instancia actual.

3.4.2 Agrupamiento de Mensajes de Encaminamiento

Cada mensaje de encaminamiento que es enviado a través de un enlace es transportado en paquetes formados por el mensaje en sí mismo y una cabecera. Existe una coherencia entre el tamaño del mensaje y la sobrecarga del paquete; cuanto más pequeño sea el mensaje contenido en un paquete, mayor será la cantidad de sobrecarga asociada a su transmisión. Por ello, es recomendable que cada paquete contenga tanta información necesaria como le sea posible, ya que el coste de transmitir el paquete a través de un enlace es inversamente proporcional al tamaño del mensaje.
Esta observación tiene que ser tenida en cuenta cuando se envían paquetes LS Update y LS Ack.

Cuando se transmite un LSA a un vecino, el router transmisor espera una confirmación para dicho LSA. Si ésta no se recibe dentro de un intervalo predefinido (definido en 5 segundos [OSPFv2]), el LSA será retransmitido. Si se recibe la confirmación dentro de este intervalo de tiempo, la retransmisión del LSA se evitará.
Esta característica puede ser explotada por los routers que, habiendo recibido un LSA, en lugar de responder inmediatamente con un mensaje de confirmación (transportado en un paquete LS Ack), esperan un periodo de tiempo (inferior al intervalo de retransmisión del nodo origen), antes de enviar el paquete LS Ack. De este modo, varios mensajes de Ack podrían ser agrupados en un solo paquete LS Ack cuando otros LSAs hayan sido recibidos dentro de un intervalo de tiempo determinado.

El mismo modelo es utilizado para los LSAs, un router podría decidir esperar un pequeño intervalo de tiempo antes de enviar un paquete LS Update. Durante este tiempo podrían programarse nuevos LSAs para ser transmitidos y, de este modo, ser incluidos también en el paquete LS Update.

3.5 Tipos de Interfaces para las Distintas Redes

OSPF está diseñado para trabajar sobre diferentes tipos de redes, para lo que define diferentes tipos de interfaces, como son: punto a punto, broadcast, NBMA (Non-Broadcast Multiple Access) y punto a multipunto.
La extensión inalámbrica de OSPF-MANET introduce un nuevo tipo de interfaz, conocida como MANET, para su uso en éste tipo de redes, que será explicado en profundidad en la sección 4.9 dedicada a OSPF-MANET.

3.5.1 Punto a Punto

En la interfaz punto a punto no es necesario asignar direcciones IP, cuando se asignan las direcciones de interfaz, estas son modeladas como enlaces stub, donde cada router anuncia dicha conexión stub a la dirección de interfaz del otro router.

Opcionalmente, puede asignarse una subred IP a la red punto a punto; en este caso, ambos routers anunciarán un enlace stub a la subred IP, en lugar de anunciarse sus direcciones de interfaz el uno al otro.

3.5.2 Broadcast

Como su propio nombre indica, es la interfaz diseñada para trabajar en las redes broadcast, como por ejemplo Ethernet. Este tipo de interfaz asume conectividad bidireccional entre todos los routers participantes en el enlace.

Los routers OSPF en este tipo de redes implementan el modo de interfaz broadcast OSPF, permitiendo el uso de un Designated Router (DR).

3.5.2.1 Protocolo Hello

En las redes broadcast, cada router se anuncia mediante paquetes Hello multicast, lo que permite que los vecinos sean descubiertos dinámicamente. Estos paquetes contienen la visión que el router tiene sobre la identidad del Router Designado y la lista de routers cuyos paquetes Hello han sido recibidos recientemente.

3.5.3 Non Broadcast Multiple Access (NBMA)

En este modo, OSPF emula una operación sobre una red broadcast; se selecciona un router designado para la red NBMA y éste genera un LSA para la red.
El modo NBMA es el más eficiente para ejecutar OSPF sobre una red no broadcast, tanto en términos de tamaño de la base de datos LS generada, como en la cantidad de tráfico de encaminamiento producido.

Por otro lado, tiene una restricción importante: es necesario que todos los routers pertenecientes a la red NBMA sean capaces de comunicarse de manera directa.

3.5.3.1 Protocolo Hello

En las redes NBMA puede necesitarse cierta información de configuración para que el protocolo Hello funcione correctamente. Cada router que potencialmente pudiera convertirse en Router Designado tendrá una lista de todos los demás routers pertenecientes a la red.
Un router con potencial para ser Router Designado enviará paquetes Hello al resto de Routers Designados potenciales en el momento en que su interfaz con la red NBMA este operativa por primera vez, lo que, al fin y al cabo, es un intento de descubrir el Router Designado para la red.
Si el propio router es elegido como tal, comenzará a enviar paquetes Hello al resto de routers pertenecientes a la red.

3.5.4 Punto a Multipunto

En redes en las que un router sólo puede alcanzar un subconjunto del total de routers de la red, la interfaz punto a multipunto puede convertirse en la mejor elección. Un router, utilizando este tipo de interfaz, describe a sus vecinos (por ejemplo, el enlace entre el router local y el vecino) como enlaces punto a punto, por lo general, definiendo la métrica del enlace en modo *por vecino* (per-neighbor).

Una interfaz punto a multipunto puede ser vista como una interfaz que mantiene una colección de enlaces punto a punto.

En una red punto a multipunto no se elige un DR ya que el "mecanismo DR" (descrito más adelante) asume la capacidad de broadcast en el enlace. A cambio, todos los routers forman adyacencias con todos sus vecinos.

3.5.4.1 Protocolo Hello

En las redes punto a multipunto, el protocolo Hello funciona ligeramente diferente a redes como las broadcast y NBMA: el paquete Hello es recibido únicamente por los vecinos con los que el router emisor se puede comunicar directamente.
En casos especiales, como en las redes malladas, podría incluir a todos los nodos de la red, en cuyo caso funciona como en una red broadcast. El paquete Hello se envía regularmente a cada vecino conectado.

Todos los routers conectados a una red punto a multipunto siempre se vuelven adyacentes, decisión que se toma cuando se establece conectividad bidireccional con un vecino.

3.5.5 Estados de las interfaces

A continuación, se describirán los diferentes estados por los que pueden pasar las interfaces en el protocolo OSPF, listados por orden de progreso en su funcionalidad, desde la más baja a la más alta.

3.5.5.1 Caída

Es el estado inicial de la interfaz, en el que los protocolos de nivel más bajo indican que la interfaz esta inoperativa, no se recibirá ni enviará ningún tipo de tráfico de encaminamiento a través de la misma.

En este estado, los parámetros de la interfaz deberían ser establecidos a sus valores iniciales, todos los temporizadores deshabilitados y no debería haber adyacencias asociadas a ésta.

3.5.5.2 Loopback

En este estado, como su propio nombre indica, la interfaz a la red se encuentra en modo loopback, por lo que se encontrará incapacitada para tratar con el tráfico normal de datos.
A pesar de esto, puede ser necesario obtener información acerca de la calidad de la interfaz, lo que puede conseguirse mediante el envío de pings ICMP, o un test de error de bits; por esta razón, es posible direccionar paquetes IP a una interfaz en estado de loopback.
Para facilitar esta tarea, este tipo de interfaces son anunciados en los router LSAs como rutas a un solo host (single host routes), cuyo destino es la dirección de interfaz IP.

3.5.5.3 En Espera

En este estado, el router trata de determinar la identidad del Router Designado o de Backup; para lograrlo, controla todos los paquetes Hello que recibe.

El router no tiene permitido elegir un Router Designado o de Backup hasta que sobrepase el estado de espera, con lo que se consigue evitar cambios innecesarios del Router Designado o de Backup.

3.5.5.4 Punto a Punto

En éste estado, la interfaz se encuentra operativa, y se conecta a una red punto a punto física o a un enlace virtual.
El router intentará formar una adyacencia con el router vecino, por lo que éste enviará paquetes Hello a su vecino cada "HelloInterval".

3.5.5.5 Otro Router Designado (DR Other)

La interfaz se encuentra en una red broadcast o NBMA, en la cual otro router ha sido seleccionado como Router Designado.
En este estado, el router no ha sido elegido como Router Designado de Backup y formará adyacencias tanto con el Router Designado como con el de Backup (si lo hubiera).

3.5.5.6 Backup

En este estado, el propio router es el Router Designado de Backup de la red a la que pertenece, y sería establecido como Router Designado si el actual fallara.
El router establecerá adyacencias con el resto de routers pertenecientes a la red.

El Router Designado de Backup realiza funciones ligeramente diferentes durante el proceso de inundación, en comparación con el Router Designado.

3.5.5.7 Router Designado

En éste estado, el propio router es el Router Designado de la red a la que pertenece.
Se establecerán adyacencias con todos los routers pertenecientes a la red, éste originará un Nnetwork-LSA para la red, que contendrá enlaces a todos los routers (incluyéndose a si mismo) pertenecientes a la red.

3.5.5.8 La Interface State Machine

Cada cambio de estado es generado mediante un evento, el cual puede provocar diferentes efectos, dependiendo del estado en que se encuentre la interfaz en ese momento.
La función de la Interface State Machine consiste, por tanto, en controlar este proceso, comprobando el estado actual de la interfaz y respondiendo, en función del evento producido, con una acción determinada [OSPFv2].

3.6 Temporizadores

Un router OSPF enviará, periódicamente, paquetes Hello para indicar su presencia actual. Además, cuando el router transmite un LSA pero no recibe confirmación dentro de un intervalo de tiempo predefinido, procederá a retransmitir de nuevo dicho LSA.
Ambos procesos dependen de temporizadores, que son esenciales en OSPF, ya que son utilizados para provocar acciones de respuesta como las descritas anteriormente.

Cuando un router transmite un paquete Hello, el demonio de encaminamiento arranca un temporizador ajustado para acabar dentro de un periodo predefinido ("HelloInterval"), que en OSPF está establecido en 10 segundos, lo que significa que el router transmitirá un paquete Hello cada 10 segundos.

Existen temporizadores equivalentes para la retransmisión de los LSAs, para determinar cuándo un vecino no ha sido escuchado desde hace tiempo, etc.

3.7 Utilización de OSPF en Redes de Gran Tamaño

La información de la topología se envía por inundación, a través de la red, en paquetes LS Update; sin embargo, la inundación pura (pure flooding) provocará, en la mayoría de los casos, retransmisiones innecesarias y gastos en los recursos de la red, como, por ejemplo, el ancho de banda.

Para limitar los efectos producidos por este sistema, se han diseñado varias propuestas: una de ellas es el "mecanismo DR" descrito en el párrafo siguiente, otra es la utilización de áreas, descritas también a continuación.

3.7.1 Router Designado

OSPF define varias optimizaciones para minimizar la sobrecarga producida por los paquetes en la red, siendo una de ellas el uso, ya mencionado, de los Routers Designados (DRs) en redes broadcast y NBMA.
Un DR es, básicamente, un router OSPF normal seleccionado para transmitir los LSAs en nombre de otros routers de la red; además, un DR también genera información en nombre de la propia red.

Cada router de la red sólo se sincronizará con el DR, propuesta diseñada para optimizar el funcionamiento en las redes broadcast, ya que, en un momento dado, sólo habrá un router generando información en nombre de la red y, al mismo tiempo, se limita el número de sincronizaciones de la base de datos de n a 1, reduciendo la cantidad de carga de encaminamiento OSPF introducida en la red.

Mencionar que el modelo DR esta diseñado únicamente para NBMA (nonbroadcast multiple access) y redes broadcast, como las redes de área local (LANs).
Ya que OSPF es utilizado comúnmente en estos escenarios, el mecanismo DR es una de las características más conocidas de este protocolo.

3.7.2 Áreas

La idea tras las áreas consiste en dividir las redes OSPF en subdominios, con el fin de reducir la inundación en la red, y limitar el tamaño de las bases de datos LS (LS database) de los routers.
Únicamente los routers dentro de un mismo área conocen la topología exacta de la misma, mientras que los routers fuera de ésta, solo reciben un resumen de la topología. De este modo, cuando se produce un cambio, inundar el área local puede ser suficiente, reduciendo la carga producida por el mecanismo de flooding (flooding overhead) en el resto de zonas del dominio de encaminamiento (routing domain).

Estas áreas son las que constituyen el Sistema Autónomo. Hay un caso especial de área, conocida como backbone, que debe existir dentro de todo AS (la figura 3.4 muestra la relación entre los conceptos de backbone y área). Todas las áreas deben conectarse directamente al backbone; éstas (incluyendo el backbone) pertenecerán al AS.

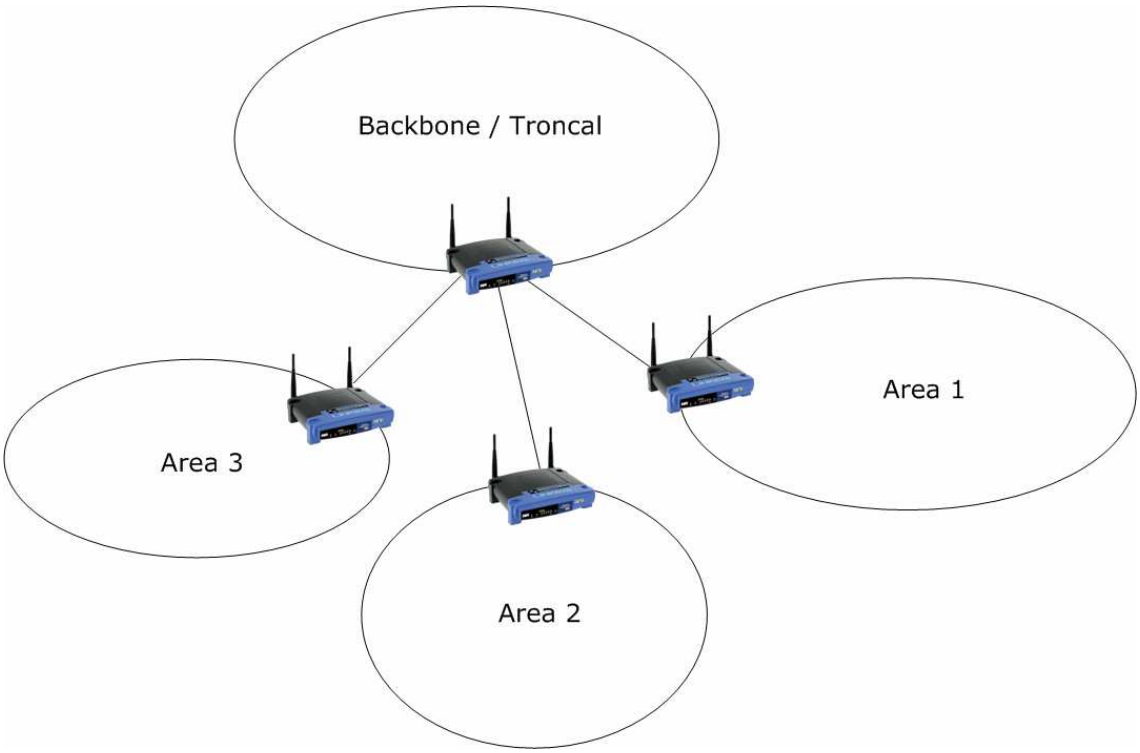


Figura 3.4 Relación entre backbone y área

Dentro de un área pueden existir diferentes redes. Como se muestra en el ejemplo de la figura 3.5, ambas redes pertenecen al mismo área.
También se puede apreciar como un router OSPF podría tener interfaces conectadas a diferentes redes. En la figura, el router A pertenece a dos redes, ambas pertenecientes a la misma área.

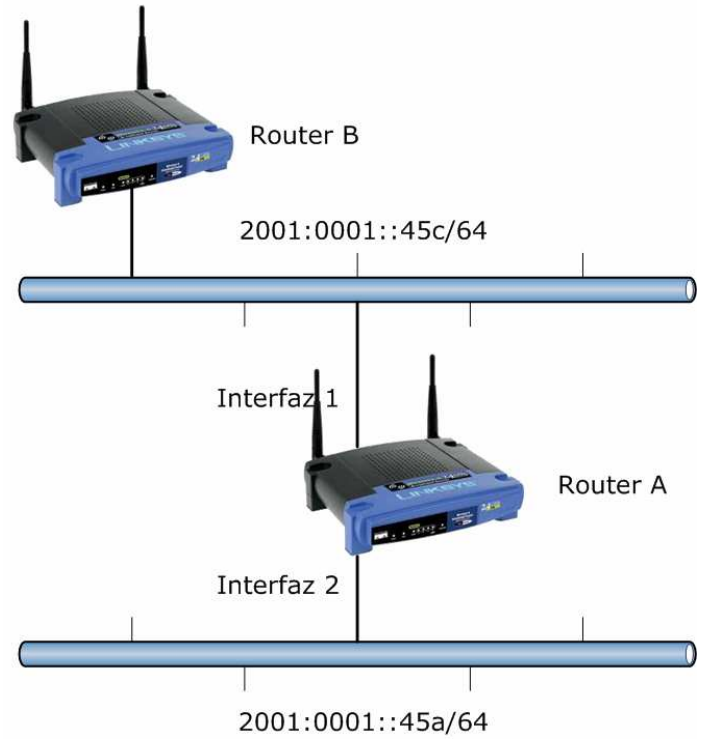


Figura 3.5 Ejemplo de router conectado a dos redes, en una misma área

3.8 Diferencias entre OSPFv2 y OSPFv3

La extensión inalámbrica descrita en la sección 3.9, y que se usará en éste estudio, se aplica a OSPFv3 [OSPFv3]. Dado que OSPFv3 y OSPFv2 son prácticamente idénticos en su funcionamiento, únicamente se mencionarán los pequeños cambios relevantes en este estudio:

Los paquetes OSPFv3, y la mayoría de los LSAs (descritos en la sección 3.4), no llevan información de direccionamiento.
OSPFv3 define dos nuevos tipos de LSAs, llamados Link LSA y Intra-Area-Prefix-LSA. El primero de ellos tiene, como su propio nombre indica, un ámbito de inundación link-local y proporciona a sus vecinos información sobre su dirección link-local (y la de otros si fuera el caso) y de su prefijo IPv6 asociado con el enlace. El segundo tipo de LSA lleva la información de todos los prefijos IPv6 que en OSPFv2 eran llevados en los Router LSAs y los Network LSAs.

Todos los routers OSPFv3 están identificados por su "router ID".

Dos routers pueden llegar a ser vecinos, a pesar de su prefijo de red asociado, debido a que se usan direcciones link-local para todas las comunicaciones entre los routers.

3.9 Extensión MANET para OSPF Mediante Inundación CDS, OSPF-MDR

Las pruebas que se realizarán en este estudio se centran, específicamente, en esta variante del protocolo OSPF, por lo que se pasa a describir su funcionamiento y características principales.

MDR es la abreviatura de MANET Designate Router [OSPFMDR], y es el concepto principal de esta extensión, la cual permite reducir la carga en redes inalámbricas, como, por ejemplo, en una red broadcast mediante la utilización de los Designate Router (DR) y Backup Designated Router (BDR), que permitirán reducir el número de adyacencias, ya que éstas sólo se establecerán entre dos vecinos si uno de ellos es el DR o BDR.

A pesar de que está diseñada originalmente para su uso en redes MANET, también es aplicable a las redes malladas inalámbricas fijas, usadas en éste estudio, mediante encaminamiento de nivel tres, como puede comprobarse en la introducción de su RFC [OSPFM].

Esta extensión, de funcionamiento proactivo, añade un nuevo tipo de interfaz, denominada MANET, para optimizar el funcionamiento sobre redes inalámbricas, bajo la condición de que la funcionalidad de las anteriores seguirá manteniéndose. Esto es debido a que las interfaces básicas existentes en OSPF no ofrecen un buen rendimiento en las redes inalámbricas, como las redes malladas, o las MANETs, a causa de:

- Los problemas generados por las operaciones de inundación a la hora de escalar la red.
- La incapacidad por parte del proceso de elección del Router Designado para converger en todos los escenarios.
- El amplio número de adyacencias que se crean cuando se utiliza un tipo de interfaz punto a multipunto.

La idea desarrollada para solucionar estos problemas consiste en generalizar los conceptos de Router Designado (DR) y Router Designado de Backup (BDR) para las redes multisalto inalámbricas, con el fin de reducir la carga minimizando el número de routers que puedan inundar LSAs y reducir el número de adyacencias. De este modo, los Routers Designados y Routers Designados de Backup generalizados pasan a denominarse Router Designado MANET (MDR) y Router Designado MANET de Backup (BMDR) respectivamente.

En OSPF, los DRs forman un connected dominating set (CDS) [CDS]; a su vez, los DRs y BDRs forman, entre ellos, un biconnected CDS (BCDS), que consiste en la formación de un backbone virtual para el encaminamiento de trafico, donde los mensajes pueden ser enviados desde el origen a un vecino, a través del CDS, hasta el miembro del *dominating set* más cercano al nodo de destino.
Este mecanismo asegura una reducción en el tráfico a través de la red; por el mismo principio, OSPF-MDR selecciona MDRs y Backup MDRs (BMDRs), así como los MDRs forman un CDS y los MDRs y BMDRs forman un BCDS.

	OSPF	**OSPF-MDR**
CDS esta formado por	DRs	MDRs
BCDS esta formado por	DRs y BDRs	MDRs y BMDRs

Tabla 3.3 Diferencias de CDS y BCDS en OSPF y OSPF-MDR

Por definición, cada router perteneciente a la red inalámbrica formará parte del CDS o estará a un salto de distancia de él.
Para seleccionar y mantener dinámicamente el BCDS, se utiliza un algoritmo distribuido: únicamente se establecerán adyacencias entre MDRs, BMDRs y un subconjunto de sus vecinos, lo que conlleva una importante reducción del número de adyacencias en redes de gran tamaño, en comparación con el sistema básico de formar adyacencias entre todas las parejas de vecinos.

Los paquetes Hello clásicos son modificados mediante el uso de señalización de enlace local (link-local signaling, LLS) [RFC5613] con dos propósitos:
- Proporcionar a los vecinos información vecinal a dos saltos (2-hop neighbor information), que especifica los vecinos bidireccionales del router, y que será utilizada por el algoritmo de selección MDR.
- Permitir Hellos Diferenciales (Differential Hellos) que solamente informen de los cambios en los estados de los vecinos.

Estos Hellos Diferenciales pueden ser enviados con mayor frecuencia, sin producir un aumento significativo en la carga de la red, para responder con mayor rapidez a los cambios en la topología.

La selección MDR es un mecanismo para decidir si un router será MDR, BMDR o "MDR Other", y se encargará de asignar el (B)MDR padre ((B)MDR parent) para todos los nodos de la red; una vez finalizado el proceso de selección, se formarán las adyacencias usando los MDRs y BMDRs establecidos.

Cada router MANET anunciará en su router-LSA un subconjunto de sus vecinos como enlaces punto a punto; la elección de los vecinos que se anunciarán es flexible, permitiendo una reducción de la carga motivada por el intercambio de menor información sobre la topología.

Los LSAs son enviados mediante inundación, desde un router (B)MDR de la red a sus vecinos. La cantidad de vecinos almacenados en el paquete router-LSA, se decide mediante un nuevo parámetro configurable "LSAFullness", por ejemplo:

- Un LSAFullness=0 significa *LSA mínimo*, y únicamente los vecinos adyacentes estarán incluidos.
- Un LSAFullness=4 significa *LSA completo,* donde todos los vecinos encaminables estarán incluidos, sumándose a los vecinos adyacentes.
- Valores entre 1 y 3 significan que sólo parte de los vecinos encaminables están incluidos. Aunque el paquete router-LSA podría no ser completo (full), el cálculo de la tabla de encaminamiento (routing table) incluye a todos los vecinos encaminables para el SPF.

Otro cambio introducido es la modificación de la Interface State Machine para manejar eventos como MDRNeighborChange. La Neighbor State Machine administra el cambio de estado de los vecinos (neighbor state change) y permite la formación de adyacencias solamente a los vecinos con estado TWOWAY o superior (visto en la sección 4.3.5.2).

3.9.1 Nueva Interfaz MANET

3.9.1.1 Campos de Datos Modificados

El campo "type" define el tipo de interfaz, y tiene cuatro tipos OSPF predefinidos (punto a punto, broadcast, punto a multipunto y NBMA). Ahora, a este grupo se le añade el tipo MANET, que permitirá identificar a la red inalámbrica y aplicarle el encaminamiento correspondiente.

El campo "state" define el estado de la interfaz; para el caso de esta extensión, el estado DR implicará estado MDR y el BDR, a su vez, BMDR.

3.9.1.2 Campos de Datos Nuevos

Además de los campos modificados, se establecen otros de nueva creación:

- MDR Level
  Tiene tres valores predefinidos: 2 para MDR, 1 para BMDR y 0 para MDR Other.

- MDR Parent
  Todo nodo que no sea MDR seleccionará un MDR padre, que, si es posible, debería ser vecino.

- Backup MDR Parent
  En el momento de establecerse una adyacencia biconectada, cada nodo que no sea MDR elegirá un BMDR padre, que, como en el caso anterior, será un MDR o BMDR vecino, si es posible.

- Router Priority
  Consiste en una variable que permite establecer la disposición de un router para ser MDR. Esta variable puede ser cambiada dinámicamente en base a diferentes criterios, como la propia disposición, la capacidad de la batería, etc.

- Hello Sequence Number (HSN)
  Es una secuencia numérica transportada por el Hello Sequence TLV (TLV consiste en un formato de codificación de tres campos; tiempo, longitud y valor).

- Lost Neighbor List (LNL)

Consiste en una lista con los Router ID de los vecinos que han cambiado su estado a caído recientemente; está incluida en la Lost Neighbor List TLV de los paquetes Hello.

3.9.1.3 Nuevos Parámetros de Interfaz Configurables

Los nuevos parámetros incluyen "2HopRefresh", "HelloRepeatCount", "AckInterval", "BackupWaitInterval", "AdjConnectivity", "MDRConstraint", "LSAFullness".
Estos parámetros ayudarán a decidir las características de los Hellos, adyacencias, LSAs y del cálculo MDR.

Capítulo 4 OLSR

4.1 Introducción

OLSR [OLSR] (Optimized Link State Routing Protocol) es un protocolo diseñado específicamente para MANETs (Mobile Ad-hoc Networks), pero también utilizable en redes malladas multisalto fijas [OLSR2]. Utiliza UDP para la comunicación de mensajes, y tiene asignado el puerto 698 para su uso exclusivo [IANA].
Basa su funcionamiento en las tablas de encaminamiento; se trata de un protocolo proactivo, lo que implica que cada nodo dispone, en todo momento, de la información necesaria para comunicarse con el resto de nodos de la red. Esto implica una velocidad de encaminamiento mayor, unido a la ventaja de conocer a todos los nodos pertenecientes a dicha red.

OLSR esta diseñado para trabajar independientemente con otros protocolos, lo que añade versatilidad a la hora de implantarlo sobre un escenario. Uno de los conceptos más interesantes que aporta OLSR es el conocido como MPR (Multipoint Relay), cuya utilidad principal es optimizar los niveles de carga de señalización introducida en la red, funcionamiento que será explicado con mayor profundidad en la sección 4.3.

El protocolo mantiene la estabilidad que aporta el uso de un algoritmo de estado de enlace (sección 4.4.1); en este caso, se trata de una optimización del protocolo de estado de enlace clásico para funcionar en redes inalámbricas.
OLSR podría optimizar la reacción a cambios en la topología, reduciendo el intervalo máximo de tiempo para la transmisión de mensajes de control periódicos. Además, ya que se mantienen continuamente rutas hacia todos los destinos, se favorecen patrones de tráfico en los que un conjunto amplio de nodos se comunican con otro conjunto similar, en el que los pares origen-destino cambien con frecuencia.
El protocolo está particularmente dirigido a redes grandes y densas, donde se saca mayor partido a la optimización obtenida mediante el uso de los MPRs. Cuanto mayor y más densa sea la red, se conseguirá un mejor comportamiento en comparación con los resultados del algoritmo de estado de enlace clásico.

OLSR está diseñado para trabajar de un modo completamente distribuido, sin depender de ningún elemento central. No requiere transmisiones de mensajes de control fiables, ya que, al enviarse periódicamente, puede aceptarse una pérdida razonable de los mismos, lo que supone una ventaja, teniendo en cuenta que estas pérdidas son frecuentes en las redes basadas en radio debido, principalmente, a colisiones o a otros problemas de transmisión.

OLSR no requiere el uso de secuencias numéricas para identificar los mensajes, puesto que cada mensaje de control contendrá ya una secuencia numérica que se incrementará para cada mensaje. Así, el receptor de un mensaje de control podrá, si es necesario, identificar fácilmente qué información es la más reciente, incluso si los mensajes han sido reordenados durante la transmisión.

Otras características interesantes que proporciona OLSR son el modo de operación de reposo y el encaminamiento multicast. Adicionalmente, permite añadir funcionalidades extras mediante extensiones o plugins, manteniendo la compatibilidad con versiones anteriores.

4.2 Mensajes OLSR

El protocolo OLSR emplea un formato único para todos sus tipos de mensajes; de este modo, la evolución del protocolo se hace muy sencilla, evitando problemas de compatibilidad con versiones más antiguas. Estos mensajes se envían, a través de la red, encapsulados en paquetes UDP y usan el puerto 698 asignado por IANA (Internet Assigned Numbers Authority).

Los paquetes pueden contener más de un mensaje, lo que reduce la carga. Un aspecto importante es la inclusión de un campo de número de secuencia, que permite a los nodos ordenar los mensajes y saber en todo momento qué información está actualizada y cuál obsoleta.

El protocolo basa su funcionamiento en la utilización de tres tipos de mensajes, resumidos en la tabla 4.1 y descritos a continuación.

Mensaje	Función
Hello	Descubrir nodos, detectar estado de enlaces, elegir MPRs
TC	Declarar la topología de la red
MID	Detectar interfaces múltiples en los nodos

Tabla 4.1 Resumen de mensajes OLSR

4.2.1 Mensaje Hello

Sus funciones principales son el descubrimiento de vecinos, la detección del estado de enlaces y la elección de los MPRs.

4.2.1.1 Formato del Mensaje

- Reserved
  El valor de éste campo debe ser "0000000000000" por motivos de diseño.

- HTime
  Especifica el intervalo de emisión de mensajes Hello utilizado por el nodo en su interfaz, o lo que es lo mismo, el tiempo que ha de pasar antes de la transmisión del siguiente Hello.

- Willingness
  Especifica la disposición de un nodo para transportar y reenviar tráfico a favor de otros nodos.

- Link Code
  Especifica información sobre el enlace entre la interfaz del emisor y las interfaces de los nodos en la lista de vecinos, además del estado del vecino.

- Link Message Size
  Indica el tamaño del mensaje de enlace, en bytes, y medido desde el comienzo del campo "Link Code" hasta el siguiente.

- Neighbor Interface Address
  Especifica la dirección de la interfaz de un nodo vecino.

4.2.2 Mensaje TC (Topology Control)

Su función principal es la declaración de topología de la red. Cada nodo almacena información de la topología, con el fin de crear sus tablas de encaminamiento.

4.2.2.1 Formato del Mensaje

- Advertised Neighbor Sequence Number (ANSN)
  Se asocia un número de secuencia al advertised neighbor set. Cada vez que un nodo detecta un cambio en su advertised neighbor set, incrementa éste número, que será enviado en este campo del mensaje TC, con el fin de hacer un seguimiento de la información más reciente.
  Cuando un nodo recibe un mensaje TC, puede decidir, en base a ésta secuencia, si la información recibida sobre los vecinos anunciados del nodo emisor es más reciente que la que tiene actualmente.

- Advertised Neighbor Main Address
  Este campo contiene la dirección principal (main address) de un nodo vecino. Todas las direcciones principales de los vecinos anunciados por el nodo emisor estarán en el mensaje TC.

- Reserved
  El valor de éste campo debe ser "0000000000000" por motivos de diseño.

4.2.2.2 Advertised Neighbor Set

El envío de los mensajes TC tiene la finalidad de declarar un conjunto de enlaces, denominado como advertised link set; este conjunto debe incluir, al menos, los enlaces a todos los nodos de su MPR Selector set (sección 4.3), es decir, los vecinos que han seleccionado al nodo emisor como MPR. El ANSN asociado con el advertised neighbor set es enviado también en dicha lista.

4.2.3 Mensajes MID (Declaración de Interfaz Múltiple, Multiple Interface Declaration)

Se encargan, como su propio nombre indica, de la detección de interfaces múltiples en los nodos. Para los nodos con una única interfaz OLSR, la relación entre la dirección de la interfaz OLSR y su correspondiente dirección principal es trivial (la dirección principal es la dirección de la interfaz OLSR). En el caso de varias interfaces OLSR, ésta relación es definida mediante el intercambio de mensajes MID.

Los nodos con interfaces múltiples deberán anunciar, periódicamente, información que describa la configuración de sus interfaces al resto de nodos. Lo que se consigue a través de la inundación de mensajes MID a todos los nodos de la red, mediante el mecanismo de inundación MPR.
Cada nodo de la red mantiene información de las interfaces del resto de nodos. Esta información es adquirida de los mensajes MID enviados por los nodos con interfaces múltiples y es utilizada para los cálculos de la tabla de rutas.

4.2.3.1 Formato del Mensaje

- OLSR Interface Address
  Este campo contiene la dirección de una interfaz OLSR del nodo, excluyendo la dirección principal (que ya está indicada en el campo "Originator Address" del paquete OLSR).

4.3 Multipoint Relays (MPRs)

La finalidad de los MPRs [OLSRMPR] es minimizar la carga producida por la inundación de mensajes en la red, reduciendo las retransmisiones redundantes en la misma zona. Cada nodo de la red selecciona un conjunto de nodos de su vecindad a un salto, con los que mantenga conectividad bidireccional (lo que evitará problemas asociados con la transmisión de datos a través de enlaces unidireccionales) a los que retransmitirá sus mensajes.
Este conjunto de nodos vecinos será el conjunto MPR de ese nodo. Los vecinos del nodo N, que no pertenecen a ese conjunto, reciben y procesan los mensajes broadcast, pero no retransmitirán este tipo de mensajes recibidos desde N.
Este conjunto de nodos es seleccionado de tal modo que cubra (en términos de alcance de la señal de radio) todos los nodos con conectividad bidireccional, estrictamente, a dos saltos. La figura 4.1 muestra un ejemplo de este mecanismo, donde los nodos "m" son los MPRs y los nodos "n" son los que están situados a dos saltos del nodo seleccionador.

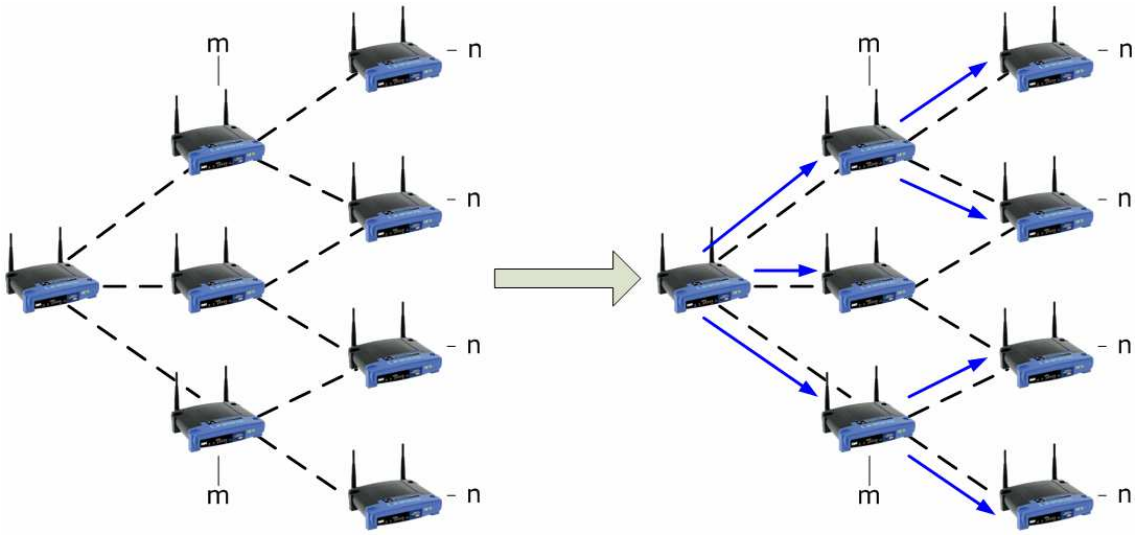


Figura 4.1 Inundación a dos saltos mediante el mecanismo MPR

El conjunto MPR del nodo N es, por lo tanto, un subconjunto arbitrario de todos los vecinos bidireccionales a un salto de N, que satisfacen la condición de que cada nodo, en la vecindad de N, con conectividad bidireccional estricta a dos saltos, debe tener un enlace bidireccional hacia el conjunto MPR de N. Cuanto más pequeño sea éste conjunto, menor carga de tráfico de control será generada por el protocolo.

Cada nodo mantiene información sobre el conjunto de vecinos que le han elegido como MPR, denominado "Multipoint Relay Selector Set" de un nodo; esta información es obtenida mediante la recepción periódica de mensajes Hello desde los vecinos. Este conjunto puede cambiar cada cierto tiempo, siendo indicado por el nodo elector en sus mensajes Hello.
Los mensajes de broadcast, provenientes de cualquiera de los "MPR selectors" del nodo N, con intención de ser difundidos por toda la red, serán retransmitidos únicamente por N.

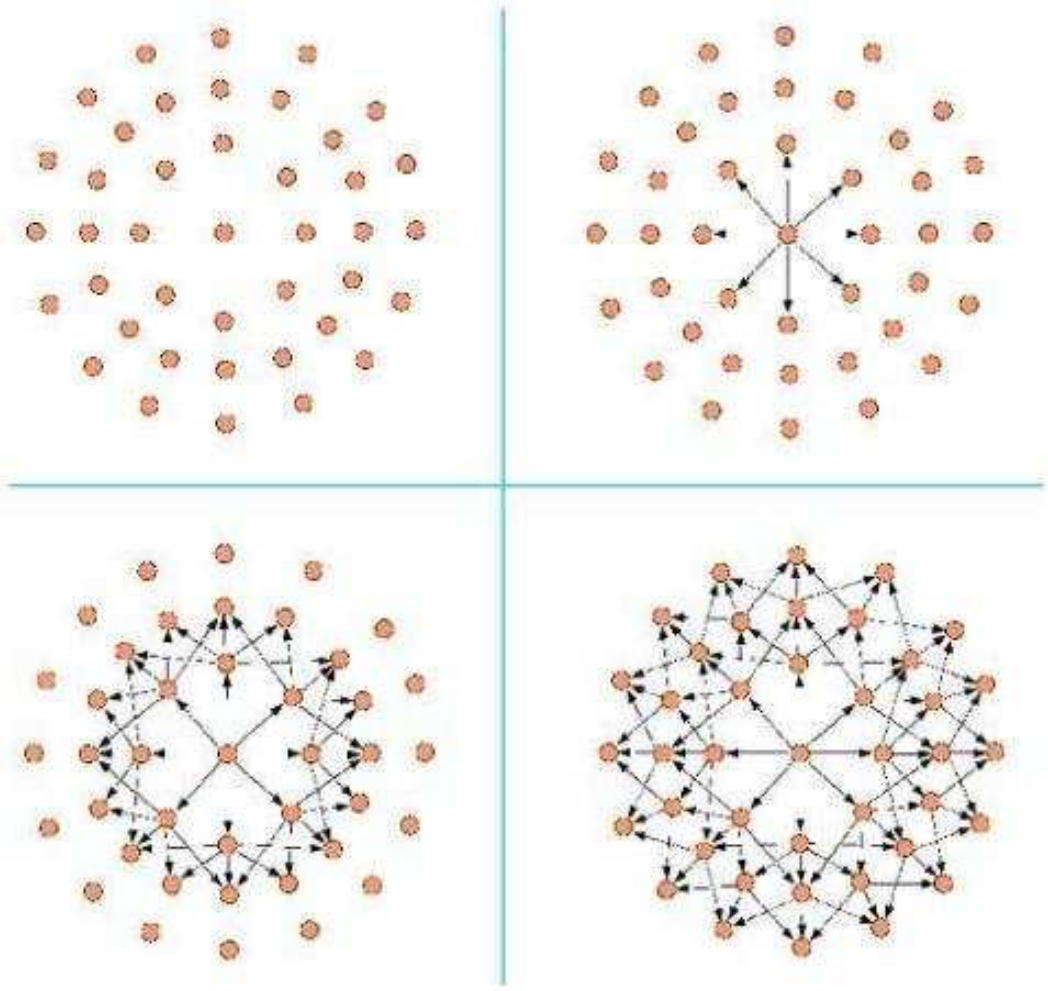


Figura 4.2 Mecanismo de broadcast clásico

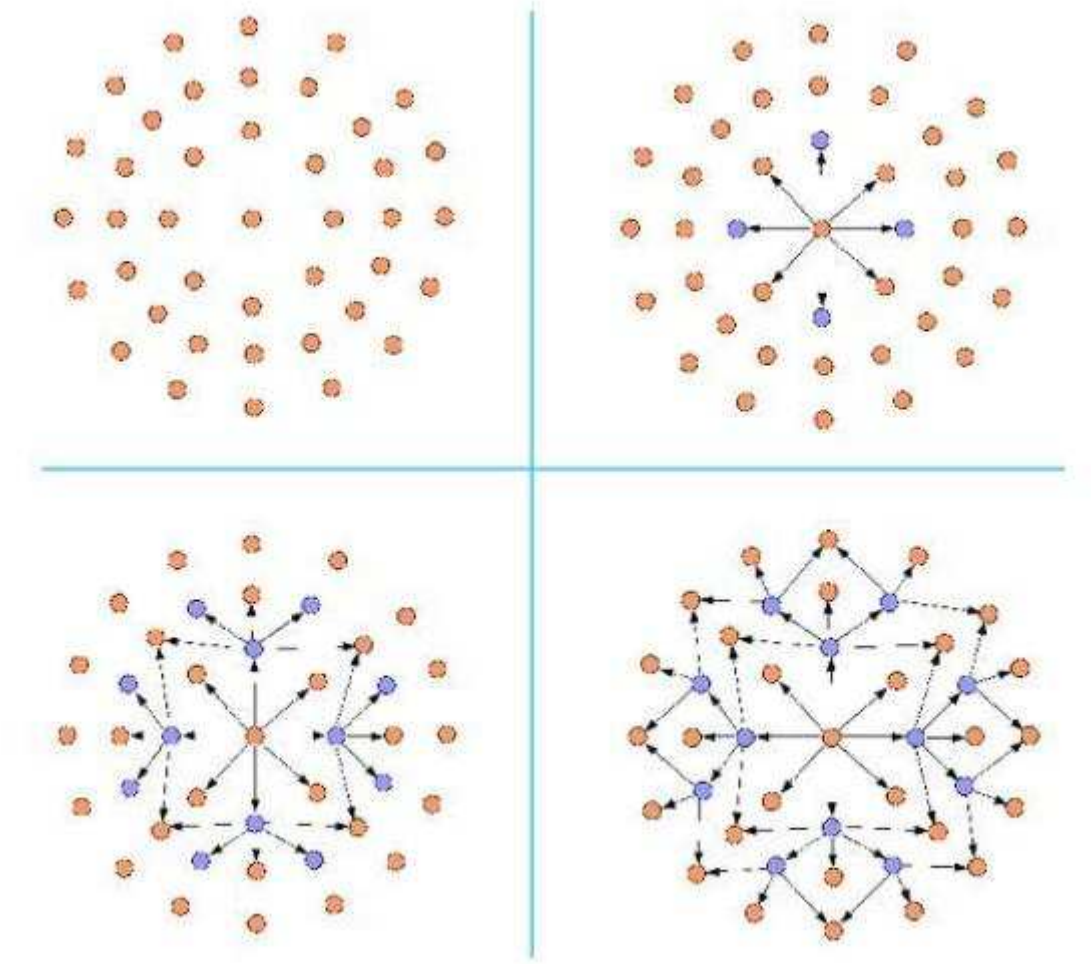


Figura 4.3 Mecanismo de inundación mediante MPRs (en azul)

De esta manera, los nodos que han sido seleccionados como MPRs difunden por toda la red la información de la topología, mediante los paquetes TC (Topology Control), utilizando el sistema descrito anteriormente.

Los paquetes TC son originados y transmitidos por los nodos MPR con la finalidad de anunciar un grupo de enlaces, conocido como *advertised link set*, entre los que han de estar, por lo menos, los enlaces a todos los nodos que lo han seleccionado como MPR.

Así, cada uno de los nodos almacena una tabla de encaminamiento, creada mediante los mensajes periódicos que recibe con la información de estado de los enlaces. Dicha tabla se va actualizando al detectar cambios en los enlaces, ya sea en el vecino, en el vecino a dos saltos, o en la topología.

Generalmente, se seleccionan rutas que atraviesan nodos MPRs, de este modo, se elimina la problemática introducida por los enlaces unidireccionales, que impiden la contestación del nodo destinatario.

Con el fin de dar mayor fiabilidad a la comunicación, se introducen MPRs redundantes, mediante la selección de más nodos MPRs de los estrictamente necesarios. Este método provoca un aumento de la carga de señalización, pero produce una conectividad más eficiente en redes grandes, asegurando que los paquetes lleguen a su destino.

4.4 Funcionamiento del protocolo

El funcionamiento del protocolo se basa en el algoritmo de estado de enlace que se describe a continuación.

4.4.1 Algoritmo de Estado de Enlace

El proceso del algoritmo [AEE] puede dividirse en cinco partes:

- Descubrir sus vecinos y aprender sus direcciones de red
  La primera tarea de un router al arrancar consiste en conocer quienes son sus vecinos, por lo que enviará un paquete Hello a través de cada enlace punto a punto. Los vecinos que reciban éste mensaje le responderán identificándose.

- Medir el retardo o el coste a cada uno de sus vecinos
  Este algoritmo necesita conocer el retardo o coste que implica la comunicación con sus vecinos, para lo que enviará un paquete especial Echo a cada uno de ellos, mediante cálculos sobre el tiempo de ida y vuelta del paquete obtendrá el coste a dicho nodo.

- Crear un mensaje donde se encuentre toda la información aprendida
  Una vez obtenida la información necesaria, se genera un mensaje que almacena toda la información que el nodo ha recogido. Este mensaje tendrá la siguiente estructura: la identidad del emisor, seguida por un número de secuencia y la edad, finalizando con la lista de los vecinos conocidos.

- Enviar dicho mensaje al resto de routers
  Esta parte es la más compleja, básicamente, consiste en transmitir el mensaje mediante inundación a través de toda la red.

- Calcular el camino más corto al resto de routers
  Tras obtenerse toda la información necesaria, el router podrá generar el grafo de red, pudiéndose utilizar el algoritmo Dijkstra [DIJKS], como ocurría en OSPF, para calcular el camino más corto a los routers de la red.

Una vez explicado el algoritmo en que se basa OLSR, se continuará con una exposición más especifica de su funcionamiento interno.

OLSR tiene una funcionalidad primaria, denominada núcleo, que siempre será necesaria para la ejecución del protocolo, y una serie de funciones auxiliares que pueden ser utilizadas o no.
Este núcleo habilita al protocolo para proporcionar encaminamiento en una red, mientras que cada función auxiliar añade funcionalidad adicional, que podría ser necesaria en escenarios específicos. Todas estas funciones auxiliares son compatibles entre si, permitiendo que cualquier subconjunto de éstas pueda ser implementado con el núcleo del protocolo, además de hacer posible que diferentes nodos, que implementen diferentes subconjuntos de éstas funciones, coexistan en la red.
El propósito de esta división entre funcionalidad principal y auxiliar está dirigido a proporcionar un protocolo simple y sencillo de comprender, añadiendo cierta complejidad solamente cuando sea requerida alguna funcionalidad auxiliar.

4.4.2 Funcionalidad Primaria

El núcleo de OLSR especifica el comportamiento de un nodo, equipado con interfaces OLSR, que participa en la red ejecutando OLSR como protocolo de encaminamiento, incluyendo una especificación universal de los mensajes OLSR y su transmisión a través de la red, así como la detección del estado de los enlaces, la difusión de topología y el cálculo de rutas.
El núcleo está formado por los componentes descritos a continuación.

4.4.2.1 Formato de Paquetes y Mecanismo de Reenvío

OLSR utiliza un formato de paquete unificado para todos los datos relacionados con el protocolo. El propósito de esta norma es facilitar la extensibilidad del protocolo sin perder la compatibilidad hacia atrás con otras versiones.
Estos paquetes se encapsulan en datagramas UDP para su transmisión a través de la red; cada uno de estos paquetes encapsulan, a su vez, uno o más mensajes, los cuales comparten un formato de cabecera común, que permite a los nodos aceptar y, si es necesario, retransmitir mensajes de tipo desconocido.

Los mensajes pueden ser inundados a lo largo de toda la red, o a un rango limitado (en términos de número de saltos) desde el nodo origen. Por lo tanto, transmitir un mensaje a la vecindad de un nodo es hablar de un caso especial de inundación. Cuando se inunda algún mensaje de control, las retransmisiones duplicadas son eliminadas localmente (cada nodo mantiene un grupo duplicado para impedir que se transmita el mismo mensaje de control dos veces) y minimizadas en la totalidad de la red mediante el uso de MPRs, como ya se describió en la sección 4.3.

4.4.2.2 Formato de Paquete

La figura 4.4 muestra la estructura básica de cualquier paquete OLSR, omitiendo las cabeceras IP y UDP.

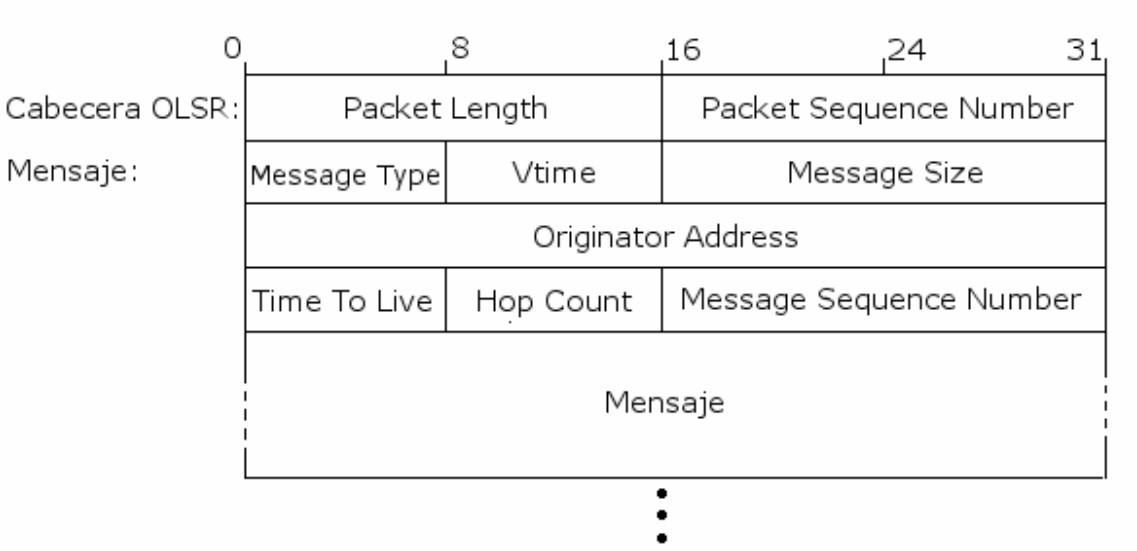


Figura 4.4 Formato de un paquete OSLR

4.4.2.3 Cabecera del paquete

- Packet Length
  El tamaño en bytes del paquete.

- Packet Secuence Number (PSN)
  Es incrementado, de uno en uno, cada vez que un paquete OLSR nuevo es transmitido, manteniéndose una secuencia diferente para cada interfaz.

La dirección IP de la interfaz, sobre la que se transmite un paquete, se obtiene de la cabecera IP del mismo.

4.4.2.4 Mensaje

- Message Type
  Indica el tipo de mensaje contenido. Los tipos de mensaje en el rango 0-127 están reservados para los ya existentes y posibles extensiones.

- Vtime
  Indica el tiempo máximo para considerar como valida la información contenida en un mensaje, tras la recepción del mismo por parte de un nodo, si no se ha recibido una actualización más reciente.

- Message Size
  Ofrece el tamaño del mensaje, en bytes, y se mide desde el comienzo del campo "Message Type" hasta el comienzo del siguiente campo "Message Type" (o, si no hubiera mensajes siguientes, hasta el final del paquete).

- Originator Address
  Contiene la dirección principal del nodo que originalmente generó el mensaje. No debe ser confundida con la dirección origen de la cabecera IP, que cambia cada vez a la dirección de la interfaz intermedia que retransmite el mensaje. Este campo nunca debe cambiar en las retransmisiones.

- Time To Live
  Este campo contiene el número máximo de saltos en los que un mensaje será transmitido. Antes de que un mensaje sea transmitido, el TTL se decrementa en 1. Cuando un nodo recibe un mensaje con un TTL igual a 0 o 1, el mensaje no se retransmitirá.
  De este modo, el originador de un mensaje puede limitar el radio de inundación.

- Hop Count
  Contiene el número de saltos realizados por el mensaje. Antes de que el mensaje se retransmita, el Hop Count es incrementado en 1.

- Message Secuence Number
  A la hora de generar mensajes, el nodo originador del mismo asignará un identificador numérico único a cada uno, que será introducido en el campo "Secuence Number" del mensaje. Esta secuencia se incrementa en 1 por cada mensaje generado por el nodo.
  Permite asegurar que un mensaje dado no será retransmitido por un nodo más de una vez.

4.4.2.5 Detección del Estado de Enlaces

La comprobación de enlaces se lleva a cabo mediante la emisión periódica de mensajes Hello, a través de las interfaces y generados individualmente para cada una de ellas, donde se comprueba la conectividad.
Por lo tanto, la detección del estado de enlaces se consigue mediante una colección de enlaces locales (local link set), que describen los enlaces entre las interfaces locales y remotas, por ejemplo, las interfaces de los nodos vecinos.

4.4.2.6 Neighbor Detection

La detección de vecinos se consigue mediante el intercambio periódico de paquetes Hello, que son enviados por los nodos, a través de la red, para anunciar su existencia al resto de nodos; esto permite a los nodos detectar a sus vecinos y conocer el estado de sus enlaces con ellos (simétrico o asimétrico).

En la figura 4.5 podemos ver un ejemplo de detección de un vecino mediante la utilización de paquetes Hello. Dichos paquetes son transmitidos periódicamente y almacenan un listado de los nodos vecinos del nodo origen.

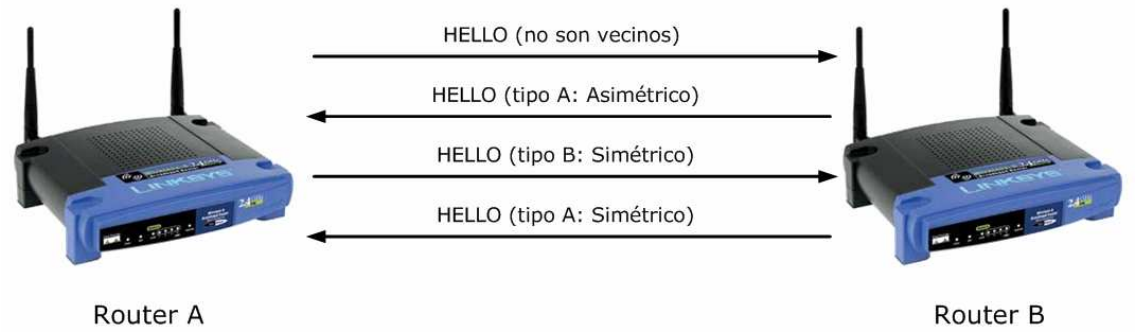


Figura 4.5 Detección de un vecino mediante paquetes Hello

El mecanismo de comprobación de enlaces, y la detección de vecinos, ofrecen a los nodos una lista de vecinos con los que comunicarse directamente y, en combinación con el formato de paquetes y el sistema de reenvío, se consigue un modo optimizado de inundación a través de los MPRs.

4.4.2.7 Selección MPR

Se sigue el mismo proceso explicado en la sección 4.3.

4.4.2.8 Difusión de Mensajes de control de la Topología (Topology Control)

Estos mensajes son difundidos con la intención de proporcionar información de estado de enlace a cada nodo de la red, para permitir el cálculo de rutas.
Con el fin de construir la base de información de la topología, los nodos que han sido seleccionados como MPR difundirán mensajes TC. Estos mensajes son inundados a todos los nodos de la red valiéndose de los MPRs, que permiten una mejor escalabilidad en la distribución de la información de la topología.

4.4.2.9 Cálculo de Rutas

Una vez se ha conseguido la información de estado de enlace, mediante los intercambios de mensajes periódicos, y la configuración de las interfaces de los nodos, se puede comenzar el cómputo de la tabla de rutas.

Cada nodo mantiene una tabla de rutas que le permite encaminar datos dirigidos a otros nodos de la red; esta tabla esta basada, como ya se ha mencionado, en la información de estado de enlace. Si se produjera algún cambio, como puede ser la caída de un enlace, se procederá a recalcular la tabla de rutas correspondiente para actualizar la información de encaminamiento de cada destino de la red.

4.4.3 Funcionalidades Auxiliares

OLSR permite la extensión de su funcionalidad mediante la utilización de plugins [OLSRDPLG1], [OLSRDPLG2], esto es, módulos independientes que permiten, tras su instalación, añadir características extra al protocolo OLSR tradicional.
Como ya se explicó anteriormente, los nuevos tipos de mensajes introducidos, si es que el plugin los introdujera, utilizarán un formato único para facilitar su compatibilidad con las diferentes versiones.

El campo "Message Type" esta preparado para soportar nuevos tipos de mensajes. Está formado por un byte, 256 valores, de los cuales 127 están reservados para mensajes propios del RFC y el resto está libre para su utilización.

Dado que la utilización de las redes malladas inalámbricas y las MANETs es reciente y se encuentra todavía en desarrollo, la posibilidad de añadir o modificar funcionalidades en sus protocolos de encaminamiento es una gran ventaja, ya que permite realizar estas tareas de un modo simple y sin alterar el funcionamiento básico del protocolo.

Algunos protocolos diseñados para redes inalámbricas, como AODV [AODV], DSR [DSR] y otros, están limitados, ya que no poseen un modo eficiente de retransmisión. El mecanismo de inundación por MPR, unido a su algoritmo de retransmisión, convierten a OLSR en una opción muy interesante.
Estas dos características permiten el funcionamiento, mediante el uso de plugins, de otras aplicaciones que requieran inundar la red como, por ejemplo, el caso del servicio de DNS; lo que quiere decir que cualquier aplicación podría utilizar el método de inundación de OLSR para enviar mensajes a través de la red.
De este modo el demonio olsrd se ejecutaría como agente de retransmisión para aplicaciones locales y así reducir la carga de la red.

OLSR permite retransmitir paquetes con cabecera OLSR, aunque se trate de un tipo desconocido de mensaje. Puede darse el caso que sólo algunos nodos de la red estén capacitados para comprender estos mensajes, pero esto no influirá en el proceso de retransmisión.
Se trata de una funcionalidad muy útil, ya que permite el empleo de la técnica de inundación de OLSR para enviar alguna información, simplemente encapsulando los datos en un mensaje con formato OLSR y añadiendo un identificador del tipo de mensaje que sea tratado por el plugin.

PARTE III
DESCRIPCION DEL TRABAJO REALIZADO

Capítulo 5 Entorno de Pruebas

5.1 Introducción

Una de las claves de este Proyecto Fin de Carrera ha sido el desarrollo, configuración y despliegue de los diferentes elementos que permitirán la creación de las redes que formarán parte del estudio, con el fin de poder analizar, de un modo exhaustivo, diferentes parámetros de red, en distintas situaciones, tanto bajo la utilización de OSPF-MANET, como de OLSR.

Por ello, el entorno de pruebas fue un factor determinante, ya que su diseño, configuración e implantación sentaron las bases para la consecución de los resultados que permitieran comparar el rendimiento de ambos protocolos.

En este capítulo se tratará sobre la utilidad del escenario de pruebas, de las motivaciones que lo definieron y de su arquitectura general.

Uno de los aspectos más importantes fue la utilización de los routers Linksys WRT54GL, dispositivos de capacidades limitadas, que posibilitó la obtención de resultados realistas y extrapolables a situaciones y escenarios reales.

5.1.1 Motivación

El número de combinaciones de los nodos a la hora de crear una topología mallada es infinita, por lo que, para simular de un modo lo más veraz posible casos representativos de modelos reales, se decidió desarrollar no uno, sino cinco escenarios aleatorios con estructura de configuración común, que únicamente se diferencian en la topología formada por los nodos que la integran; éstas estarán compuestas por un número elevado de nodos, concretamente veinte.
De este modo, se consigue representar la variedad de posibles escenarios y, mediante el uso de un número elevado de nodos, simular el comportamiento en redes de gran tamaño.

Para el desarrollo de la pruebas se utilizaron dos tipos de dispositivos: ordenadores de sobremesa y routers Linksys WRT54GL, con los que se intentó cubrir todo tipo de situaciones como aumento del número de nodos, fallos de nodos, fallos de enlaces, etc.

5.1.2 Escenario de Aplicación

El escenario de aplicación que se intenta simular consiste en una red mallada inalámbrica, formada por un número elevado de nodos, donde cada uno de ellos dispondrá de una interfaz inalámbrica configurada en el mismo canal de radio.

Teniendo en cuenta la necesidad de comunicación, será necesario que cada nodo disponga de una dirección IP única y se utilice un protocolo de encaminamiento. Este último aspecto será la finalidad de las pruebas; concretamente, analizar el comportamiento de los dos protocolos de encaminamiento, OSPF-MANET y OLSR, en dicho escenario de aplicación.

5.2 Arquitectura de la Red

Esta sección describirá en detalle la arquitectura utilizada en la red, por lo que será necesario introducir tanto los diferentes dispositivos utilizados para la realización de las pruebas, como el software empleado en ellos, además de las herramientas adicionales que fueron utilizadas.

La arquitectura adoptada durante el desarrollo de las pruebas de OSPF-MANET y OSLR está constituida por una red mallada inalámbrica, cuyas características específicas para las pruebas se citan a continuación:

- Redes aisladas

La red creada no tendrá configuración de acceso a Internet, se tratará simplemente de una red privada, puesto que este aspecto no forma parte del estudio.

- Red multisalto

El uso de una red mallada inalámbrica implica un escenario multisalto, donde las rutas entre los distintos nodos de la red pueden estar formadas por un número variable de saltos.

5.2.1 Descripción de la Arquitectura de Red

A partir de estas características básicas y ajustándolas a la estructura genérica de los escenarios de prueba que se desarrollarían, se compuso una estructura de red cuyo esquema puede verse en la figura 5.2.

A continuación se explicarán las diferentes partes que constituyen el escenario en cuestión; en primer lugar los dispositivos hardware que la forman, después las redes que se utilizaron y, finalmente, los programas y herramientas software empleadas.

5.2.1.1 Dispositivos Hardware

- Ordenadores de sobremesa

Durante las pruebas se utilizaron dos ordenadores de sobremesa. Ambos disponían de dos interfaces de red cableada (eth0 y eth1) y en uno de ellos se instaló una interfaz inalámbrica (wlan0), que sería utilizada para la captura de tráfico en la red mallada inalámbrica.
En un primer momento, sirvieron como banco de pruebas del hardware y el software que se utilizaría en los escenarios de prueba seleccionados; este proceso será desarrollado más profundamente en el capítulo 6.
Una vez concluido el banco de pruebas, uno de los PCs fue utilizado para la ejecución de los diferentes scripts de configuración y pruebas sobre los escenarios.

- Routers Linksys WRT54GL

Los routers empleados en las pruebas son Linksys WRT54GL versión 1.1 [WRT54], que utilizan el chip Broadcom 5352 (compatible con la serie 47xx).
Disponen de 4MB de memoria flash y 16MB de RAM, poseen un switch de cinco puertos integrado en la CPU, una interfaz inalámbrica y están basados en Linux.

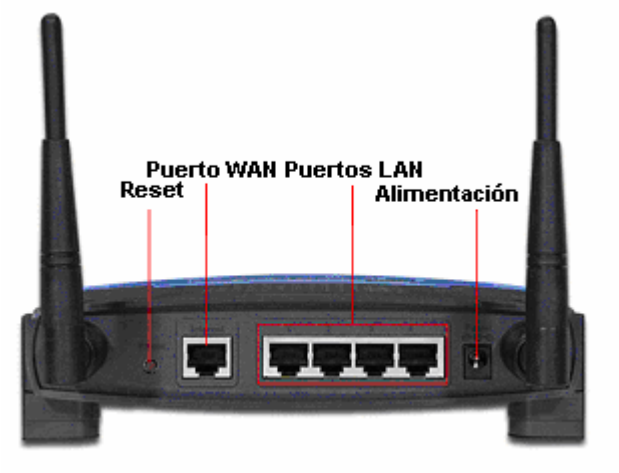


Figura 5.1 Conexionado del router Linksys WRT54GL

Debido a las obligaciones de la GNU GPL, estos routers disponen de un código fuente del firmware libre, lo que permite modificarlo para añadir o cambiar funcionalidades en el dispositivo. Esta versatilidad a la hora de disponer de un firmware programable permite cambiar el firmware original por otro más adecuado a las tareas que se realizarán durante las pruebas, en concreto Openwrt, cuyas características podrán verse con más detalle en la sección 5.2.1.2.

Por defecto, la configuración de este modelo de router es la siguiente:

Nombre de Interfaz	Descripción	Valor por defecto
br-lan	LAN & WiFi	192.168.1.1/24
vlan0 (eth0.0)	Puertos LAN (1 al 4)	Ninguno
vlan1 (eth0.1)	Puerto WAN	DHCP
wl0	WiFi	Deshabilitado

Tabla 5.1 Configuración por defecto de las interfaces

Esta configuración no se ajusta a las necesidades de las pruebas, por lo que habrá de ser modificada, según se cita en el apéndice D.

5.2.1.2 Configuración de la Red

En la figura 5.2 puede observarse como todos los dispositivos que forman parte de la red emplean uno de sus interfaces (vlan0 en el caso de los routers Linksys y eth1 en el ordenador) para conectarse a la red cableada; se trata de una conexión para la gestión de la red, que se utiliza para la configuración de los dispositivos durante la ejecución de los scripts y la obtención de los ficheros de salida, a partir de los cuales se extraerán los resultados.
Para llevar a cabo estas tareas de gestión se utiliza la subred 192.168.2.0/24; de esta manera todos los dispositivos quedan conectados entre sí por una interfaz cableada.

Debido a que la configuración por defecto de las interfaces de los nodos no se adecua a las necesidades requeridas durante las pruebas, se hace obligatoria su reconfiguración, como se detalla a continuación.

Nombre de Interfaz	Descripción	Valor por defecto
vlan0 (eth0.0)	Puertos LAN (1 a 3)	192.168.2.X/24
vlan1 (eth0.1)	Puerto LAN 4	192.168.200.X/24
vlan2 (eth0.2)	Puerto WAN	DHCP
wlan0	WiFi	192.168.3.X/24

Tabla 5.2 Configuración utilizada en las pruebas

Como se puede ver en la tabla 5.2, se ha eliminado el bridge que por defecto existía entre la interfaz cableada y la inalámbrica, asignándolas a dos subredes diferentes, ya que, por motivos de eficiencia a la hora de realizar las diferentes pruebas, las tareas de configuración y ejecución de scripts se realizarán únicamente por la interfaz cableada; de este modo, dejamos la interfaz inalámbrica libre, para que únicamente sea utilizada por los protocolos de encaminamiento. El proceso completo de cómo realizar estas modificaciones puede verse en el apéndice D.

Además, la interfaz cableada se ha separado en tres subredes diferentes, la 192.168.2.0/24, que será utilizada para configuración y en la ejecución de los scripts, la 192.168.200.0/24, libre por si fuera necesaria por algún motivo y, por último, una subred autoconfigurable por DHCP (Dynamic Host Configuration Protocol).

Como puede observarse en la figura 5.2, para las pruebas como tales, la interfaz inalámbrica de cada uno de los routers estará asociada a una subred IPv6 diferente, siguiendo un esquema de asignación de la dirección IPv6 global en formato 3ffe:X::1/128, donde X corresponderá al identificador asignado al router en cada caso; por ejemplo, para el Router 1, X valdrá 1.

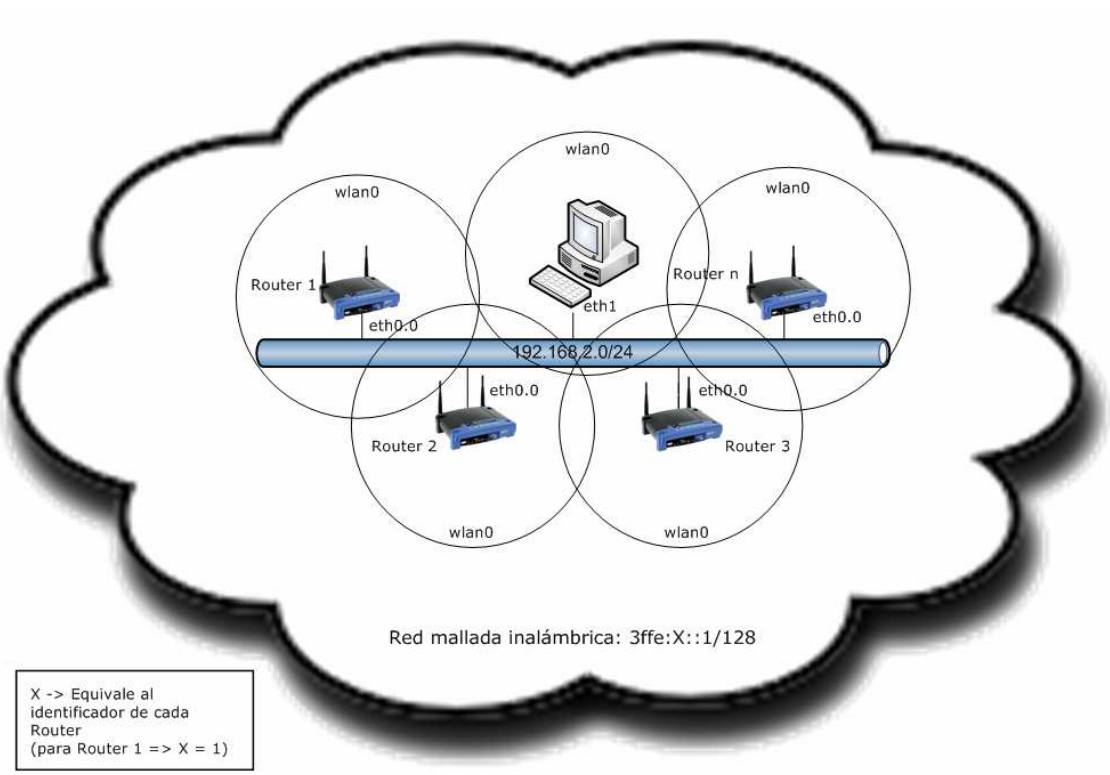


Figura 5.2 Esquema del entorno de pruebas

5.2.1.3 Software

Para los ordenadores de sobremesa, el sistema operativo seleccionado fue Ubuntu, ya que se trata de una de las distribuciones gratuitas basadas en Linux más conocidas y utilizadas. La versión instalada en ambos PCs fue la Desktop 8.04 LTS [UBUNTU] (Long Time Support) para procesadores Intel x86 de 32 bits, también conocida como "Hardy Heron", que era la última versión con soporte extendido hasta la fecha.

En el caso de los routers Linksys WRT54GL [WRT54], se instaló el firmware Openwrt [OPENWRT] (apéndice D), que consiste en una distribución Linux para dispositivos embebidos. En vez de crearse un firmware estático, ofrece una estructura de ficheros con gestión de paquetes modificable, lo que permite elegir y configurar aplicaciones de diferentes fabricantes.
Desde el punto de vista del desarrollador, Openwrt permite crear una aplicación sin la necesidad de orientar todo el firmware alrededor de la misma; para un usuario, significa la capacidad total de configuración de cada uno de los elementos que lo componen.

Primeramente, trató de instalarse una versión estándar del firmware Openwrt, disponible en su web, pero el limitado espacio de almacenamiento impedía la instalación de todas las aplicaciones y herramientas necesarias para la ejecución de las pruebas, por lo que fue necesario crear una versión del firmware personalizada, con la funcionalidad mínima imprescindible para la correcta realización de las pruebas y garantizando el espacio de almacenamiento necesario para las aplicaciones; los detalles sobre la realización de esta tarea pueden verse en el apéndice C.

A continuación se citan las diferentes herramientas que fueron necesarias para la realización de este estudio:

- Herramientas de configuración

Las redes que se desplegaron durante las pruebas fueron IPv4, para la red de configuración, e IPv6, para las redes de prueba de protocolos; por lo tanto, eran necesarias utilidades que soportaran ambos tipos de direccionamiento.
Los comandos *ip* e *iwconfig* permiten configurar las interfaces de red, tanto cableadas, como inalámbricas, de una manera muy completa.
El comando *iptables* [IPTABLES] permite definir políticas de filtrado del tráfico que circula por la red, lo que permitirá simular diferentes topologías, descartando o aceptando paquetes provenientes de los diferentes nodos de la red.
De este modo se facilita enormemente la realización de las pruebas, en las que intervienen numerosos nodos equipados con interfaces inalámbricas, cuyo despliegue real para establecer los distintos escenarios sería inviable por motivos de espacio y manejo.

- Herramientas de gestión y conexión remota

Los comandos *scp* y *ssh* ofrecen la posibilidad de acceder remotamente a otros equipos, lo que es imprescindible a la hora de realizar las pruebas en los routers desde los PCs. Por otro lado, el amplio número de dispositivos hardware utilizados durante las pruebas hace necesaria la utilización de scripts de configuración y ejecución para la automatización de las tareas; para ello, se desarrollaron programas escritos en *shell script*.

- Herramientas de captura de tráfico

El programa *tshark* [TSHARK] es un analizador de protocolos de red mediante línea de comandos, permite capturar paquetes de datos en tiempo real en una red o la

lectura de paquetes desde una captura previamente salvada. Usa el formato de captura de *libpcap* o *tcpdump* [TCPDUMP].
Esta utilidad servirá para la captura y el análisis de mensajes de señalización durante las pruebas realizadas mediante scripts, mientras que el programa *capinfos* [CAPINF] permite la obtención de datos estadísticos a partir de ficheros de captura de paquetes previamente salvados; en este caso, se utilizarán, como entrada del programa, los ficheros generados por el programa *tshark*.

- Implementación del protocolo OSPF-MANET

Se utilizará para ello el software de enrutamiento Quagga [QUAGGA], que permite convertir cualquier máquina que disponga de un sistema operativo basado en Linux en un router.
Como tal, este software no ofrece la funcionalidad requerida para este estudio, ya que sólo dispone de funcionamiento OSPF básico, por lo que será necesario aplicar el parche *OSPFv3 MANET MDR* (2 de Junio de 2008) [PARCHEMDR], para conseguirla.
El proceso completo de cómo parchear el software puede verse en el apéndice B.

- Implementación del protocolo OLSR

Se utilizará el demonio olsrd [OLSRD], existente como paquete compilado para el entorno Openwrt. El proceso completo seguido para la instalación y configuración de OSPF-MANET y OLSR puede verse en el apéndice E.

5.3 Conclusiones

En este capítulo se ha descrito la arquitectura de red utilizada durante el desarrollo de las pruebas, incluyendo sus necesidades hardware y software, lo que sirve de introducción para los siguientes capítulos, en los que se describirán los escenarios, las pruebas realizadas y los resultados obtenidos.

La figura 5.2, muestra el esquema genérico de los escenarios de prueba, la única diferencia incorporada por cada uno de los escenarios será el uso de una topología inalámbrica diferente para cada caso.

Hay que destacar que, en un primer momento, se utilizaron los ordenadores de sobremesa tanto para comprobar el correcto funcionamiento de las distintas herramientas, como para realizar una prueba previa de funcionamiento básico de los protocolos OSPF-MANET y OLSR, instalando ambas implementaciones en los PCs.
Además, antes del despliegue de las topologías aleatorias, se desarrolló un escenario con una topología determinista con un número menor de nodos, sobre la que se realizó una primera batería de pruebas con OSPF-MANET y OLSR.

Capítulo 6 Escenarios y Batería de pruebas

6.1 Introducción

En este capítulo se explicarán todos los escenarios que han sido utilizados durante la realización de los experimentos y la batería de pruebas que se llevó a cabo sobre los mismos, para analizar el comportamiento de los protocolos OSPF-MANET y OLSR.

Primeramente, se describirán las pruebas, comenzando con la topología determinista (sección 6.2.1). La cual sirvió, entre otros fines, para comprobar el correcto funcionamiento del software en el entorno de pruebas, antes de su implantación en los escenarios con topologías aleatorias, así como para obtener los primeros resultados tanto para OSPF-MANET como OLSR.

Finalmente, se presentarán las topologías aleatorias (sección 6.2.2), junto con la batería de pruebas que se realizó sobre ellas.

6.2 Descripción de las Pruebas Realizadas

Antes de empezar a describir los diferentes escenarios de topología aleatoria, se describirán las configuraciones básicas que se probaron antes de comenzar las mediciones, comprobando la funcionalidad básica de las implementaciones de OSPF-MANET y OLSR, tanto en los PCs como en la topología determinista.

Esta sección se dividirá en dos partes: una primera de pruebas de funcionamiento básico (sección 6.2.1), donde se comprobará que tanto el software, como las implementaciones de OSPF-MANET y OLSR funcionan correctamente sobre el entorno de pruebas y una segunda sobre las topologías aleatorias (sección 6.2.2), donde se describirán los escenarios usados y las pruebas realizadas sobre los mismos.

En este apartado, se tratarán de explicar las diferentes pruebas realizadas desde un punto de vista funcional, sin entrar en detalle de protocolos ni algoritmos complejos.
La información exhaustiva referente a estas pruebas puede encontrarse en el apéndice A.

6.2.1 Pruebas de Funcionamiento Básico

El primer paso consistió en instalar el sistema operativo Linux en los dos PCs, siguiendo con la implementación de los protocolos y las aplicaciones que serían utilizadas durante los experimentos, con el fin de comprobar su correcto funcionamiento.
Ambos PCs contaban con procesadores Intel x86 de 32 bits, 1 Gigabyte de memoria RAM y 2 tarjetas de red Ethernet 10/100 Mbps. En uno de ellos se instaló una tarjeta inalámbrica necesaria para las capturas de tráfico durante las pruebas.

Dado que estas pruebas sólo tratan de comprobar el correcto funcionamiento del software empleado durante los experimentos, se usaron únicamente las interfaces cableadas antes mencionadas, dejando las pruebas en interfaces inalámbricas para ser realizadas, en su momento, en los routers.

Todas las comprobaciones fueron satisfactorias. El proceso detallado que se siguió puede verse en el apéndice A; algunos detalles de interés, sobre este aspecto, serán desarrollados en el apartado dedicado a las pruebas en los routers.

El siguiente paso consistió en la ejecución de la batería de pruebas sobre la topología determinista; para ello fue necesario, primeramente, el diseño de la misma. Este escenario estaba formado por una topología simple en rombo compuesta por nueve nodos que debía de cumplir las siguientes características:

- No ser demasiado compleja.
- Utilizar todos los nodos, en este caso nueve, y no contar con ningún nodo inaccesible a la red.
- Ofrecer la posibilidad de generar rutas alternativas, en caso de algún fallo en un nodo o enlace.

Cumpliendo estos requisitos, la topología seleccionada fue la siguiente:

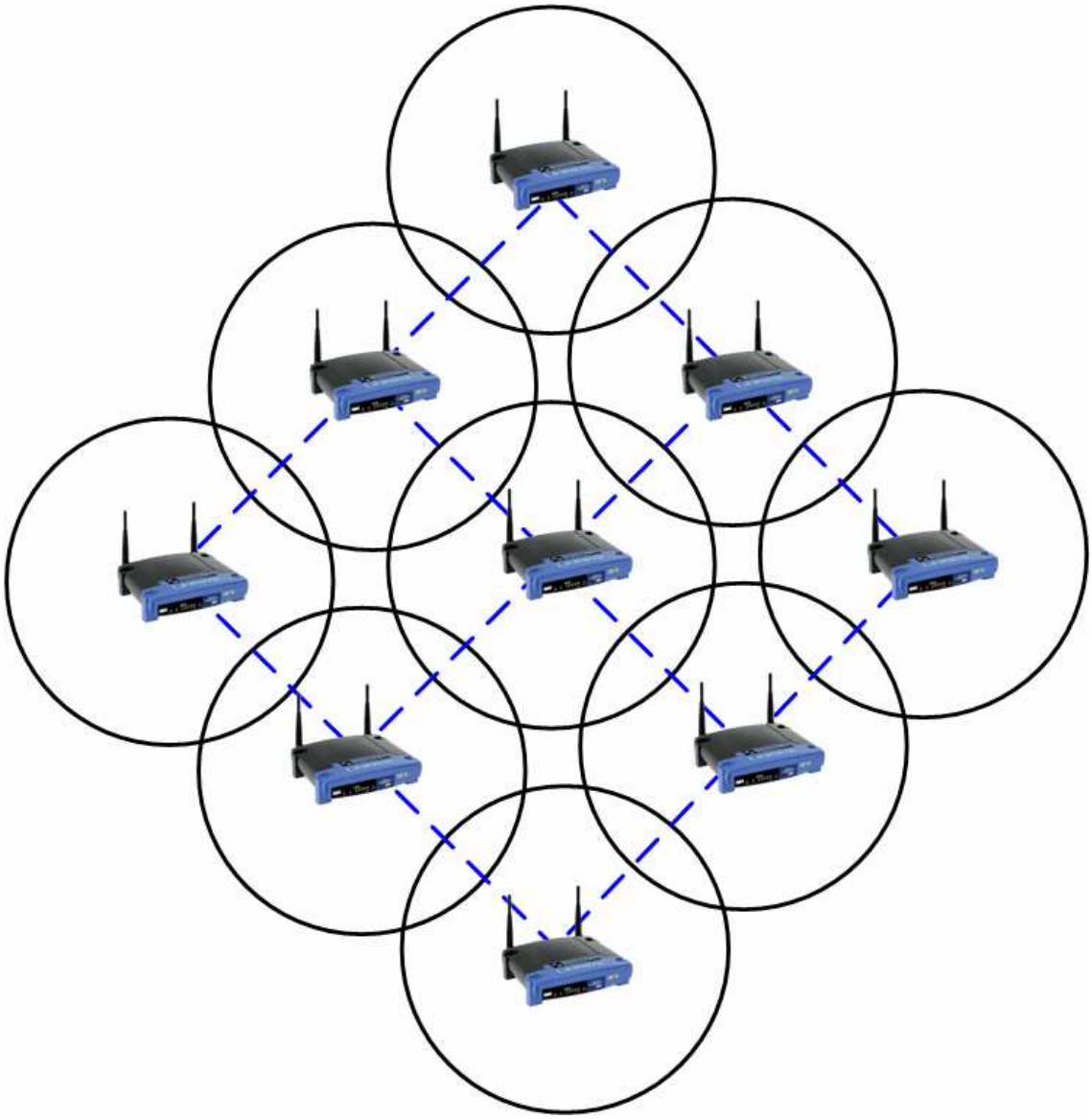


Figura 6.1 Topología empleada en la prueba con nueve nodos

Una vez decidido el escenario con topología determinista, el siguiente paso consistió en el desarrollo de la batería de pruebas que se ejecutarían sobre el mismo y sobre los escenarios con topología aleatoria.

La idea principal para el desarrollo de la batería de pruebas fue crear, primeramente, un único script que aportara una base de trabajo a partir de la cual desarrollar todas las medidas deseadas, minimizando de este modo el trabajo, dado el elevado número de medidas que se tomarán.
Por lo tanto, la finalidad de este script (apéndice F) será agrupar las tareas de configuración comunes a todas las pruebas que formarán la batería. Estas tareas consistirán en la asignación de una dirección IPv6 global a las interfaces inalámbricas de cada nodo y en el establecimiento de una topología entre los nodos mediante *ip6tables*.
Esta será, en principio, la base de todos los scripts; a partir de aquí, los scripts desarrollados para cada prueba de la batería utilizarán el script base añadiéndole la funcionalidad adicional correspondiente a su caso particular, proceso que será explicado con profundidad en la sección 6.2.2.

Dado el elevado número de escenarios sobre los que se desarrollarían las pruebas y el elevado número de éstas, se consideró necesario que los scripts desarrollados se adaptaran automáticamente a la topología sobre la que fueran ejecutados para evitar, de este modo, que cualquier cambio en la misma o en los nodos obligue a modificar manualmente todo el script para adaptarlo a los mismos.
Como se menciono anteriormente, sin este diseño, el elevado número de escenarios y pruebas hacen poco recomendable su modificación manual para cada caso, tanto por el trabajo que ello supone, como por la elevada probabilidad de cometer errores; por ello, se decide realizar el diseño de modo que la información de la topología y los nodos sea obtenida por cada script de un fichero de configuración externo, que será el que se modifique dejando el script intacto, agilizando de esta manera el proceso.

Este fichero, llamado *nodos* (apéndice F), contiene la información de todos los nodos que formarán parte de la topología: dirección IP, dirección MAC, coordenada *X* en el espacio, coordenada *Y* en el espacio y el radio de alcance inalámbrico.
La obtención de la topología a partir de estos datos se basa en el cálculo de las distancias entre los routers a partir de sus coordenadas *X* e *Y*, junto con su radio de alcance.
A todos los nodos que se encuentren a un salto (tengan visión directa) entre sí, se les permitirá reenviarse paquetes mediante *ip6tables*; a los nodos que, al contrario, no tengan visión directa, se les impedirá el reenvío directo de paquetes entre ellos, teniendo que reenviarlos salto a salto hasta alcanzar el destino.

Resumiendo, la funcionalidad principal que debían garantizar los scripts era:

- Ofrecer una base de trabajo adaptable a las diferentes configuraciones.

- Automatizar el proceso de obtención de topologías mediante el uso de un fichero de entrada externo para evitar cambios en los scripts en cada caso.

Gracias a esto, será posible desarrollar una única batería de pruebas que pueda ser ejecutada en cada uno de los escenarios de prueba, mediante una modificación simple del fichero de entrada con la información de la topología correspondiente.

Puesto que la batería de pruebas es común para todos los escenarios, y para no ser reiterativo, el desarrollo particular del funcionamiento y las medidas tomadas de cada uno de los scripts se realizará en la sección 6.2.2, dedicada a los escenarios con topología aleatoria.

Los resultados de este escenario, así como de los escenarios con topología aleatoria, serán descritos en el capítulo 7, dedicado a los resultados de los diferentes escenarios.

6.2.2 Batería de Pruebas y Escenarios con Topología Aleatoria

Las pruebas de funcionalidad básica y el escenario desarrollados en la sección anterior han servido para establecer las bases de los escenarios con topología aleatoria, que permitieron evaluar el comportamiento de los protocolos OSPF-MANET y OLSR.

A continuación, se describirán tanto las medidas de interés que se tomaron durante las pruebas, como el funcionamiento de cada uno de los scripts que componen la batería de pruebas y mediante los cuales se realizaron dichas mediciones.

6.2.2.1 Tiempo de Recuperación ante Caída de Nodo

Tiempo que tardan en generarse rutas alternativas entre los diferentes nodos afectados, a partir del momento en el que uno de los nodos que intervenía en la ruta original, que se ve afectada, deja de estar activo.
Se tomarán al menos ochenta muestras.

El funcionamiento del script (apéndice F) encargado de esta medición es el siguiente:

1. Recibe como parámetros la IPv4 del nodo que se desea tirar y el radio de alcance inalámbrico de los nodos que forman la topología, en metros; este último es opcional (su valor por defecto es de 100 metros).

2. Extrae la información del fichero de entrada "nodos" (cuya utilidad ya fue descrita en la sección anterior) para poder utilizarla en los procesos del script.

3. Reinicia todos los routers pertenecientes a la topología para eliminar cualquier configuración anterior.

4. Procesa, uno a uno, los nodos con el fin de establecer la topología de la red. Para ello, primeramente, se le asigna al nodo actual una dirección IPv6 en base a su identificador (un número de 1 a 20); a continuación, se realiza el análisis de las coordenadas X e Y de cada nodo existente en el fichero "nodos", y se comparan las del nodo actual en base al radio de alcance inalámbrico, permitiendo conectividad directa, únicamente, con los nodos situados a un salto (visión directa). Esto se consigue mediante la utilización del comando ip6tables, estableciendo por defecto la regla de descartar todos los paquetes que no estén destinados al propio router y aceptar solamente los provenientes de los nodos a un salto.

5. Una vez terminado el establecimiento de la topología, se procede a arrancar el demonio (o demonios en el caso de OSPF-MANET) encargado de ejecutar el protocolo correspondiente a la prueba, ya sea OSPF-MANET u OLSR, en cada uno de los nodos de la red.

6. Se realizan pings entre todos los nodos de la red, durante la realización de cada uno de ellos se tirará la interfaz inalámbrica del nodo elegido mediante el comando "ifdown", y se almacenará el tiempo que tarda el ping desde que pierde la comunicación hasta que la nueva ruta alternativa se establece y la

comunicación se recupera; una vez obtenido se levantará de nuevo la interfaz inalámbrica mediante el comando "ifup", este proceso se repetirá hasta realizar todas las combinaciones posibles entre los nodos de la red.

6.2.2.2 Tiempo de Recuperación ante Caída de Enlace

Consiste en el tiempo que tardan en generarse rutas alternativas entre los diferentes nodos afectados, a partir del momento en el que uno de los enlaces pertenecientes a la ruta original que se ve afectada deja de estar activo (apéndice C.7).
Se tomarán al menos ochenta muestras.

El funcionamiento del script (apéndice F) encargado de su medida es similar al descrito en la sección anterior, con la salvedad de que, en lugar de tirar completamente la interfaz con el comando "ifdown", únicamente se tirará el enlace formado entre los dos nodos que se le pasen como parámetro (en lugar de un único nodo como en el caso anterior), utilizando para ello el comando "ip6tables" y descartando, en cada uno de los dos nodos, los paquetes provenientes del otro.

Para la recuperación del enlace, se usa la misma idea que en el script anterior, en el que se recuperaba la interfaz levantándola de nuevo con el comando "ifup"; en este caso se realiza aceptando de nuevo los paquetes entre ambos nodos.

6.2.2.3 Tiempo de Convergencia Total en Alcanzar el Régimen Estacionario

Consiste en el tiempo que tarda la topología, desde que todos los routers han sido encendidos, en alcanzar el régimen estacionario (steady state), es decir, con todas las rutas definitivas establecidas y todos los nodos alcanzables.
Se tomarán al menos sesenta muestras.

El funcionamiento del script (apéndice F) encargado de esta medición es el siguiente:

1. Extrae la información del fichero de entrada "nodos" (cuya utilidad ya fue mencionada en la sección anterior) para poderla emplear en los procesos del script.

2. Reinicia todos los routers pertenecientes a la topología para eliminar cualquier configuración anterior.

3. Una vez reiniciados todos los routers, se almacena el tiempo actual, en segundos, utilizando el comando "date". La idea es utilizar este valor como tiempo de inicio, esperar a que la topología alcance el régimen estacionario y obtener el tiempo de fin, mediante la diferencia entre ambos valores se obtendrá el tiempo total.

4. Procesa, uno a uno, los nodos con el fin de establecer la topología de la red. Para ello, primeramente, se le asigna al nodo actual una dirección IPv6 en base a su identificador (un número de 1 a 20); a continuación, se realiza el análisis de las coordenadas X e Y de cada nodo existente en el fichero "nodos" y se comparan con las del nodo actual en base al radio de alcance inalámbrico. Se permite conectividad directa, únicamente, con los nodos situados a un salto (visión directa), lo que se consigue mediante la utilización del comando ip6tables, estableciendo por defecto la regla de descartar todos los paquetes que no estén destinados al propio router y aceptar solamente los provenientes de los nodos a un salto.

5. Una vez terminado el establecimiento de la topología, se procede a arrancar el demonio (o demonios en el caso de OSPF-MANET) encargado de ejecutar el protocolo correspondiente a la prueba, ya sea OSPF-MANET u OLSR, en cada uno de los nodos de la red.

6. Se espera a que la topología alcance el régimen estacionario, lo que se consigue mediante la comprobación continua de la lista de nodos alcanzables de cada uno de los nodos de la red, hasta que su número de elementos coincida con el número de nodos pertenecientes a la red.

7. En cuanto se alcanza el régimen estacionario, se almacena el tiempo en segundos, que será considerado el tiempo de fin; el tiempo que se tarda en alcanzar el régimen estacionario se calcula por diferencia del tiempo de fin menos el tiempo de inicio.

6.2.2.4 Tiempo de Convergencia de un Nodo Nuevo

Consiste en el tiempo que tarda la red en añadir, de una manera estable, un nuevo nodo, estableciéndose todas las rutas pertinentes al resto de nodos de la topología. Se tomarán al menos sesenta muestras.

En este caso, la idea consiste en seleccionar un nodo situado en uno de los extremos de la topología, que inicialmente no formará parte de la misma.
A continuación, se esperará a alcanzar el régimen estacionario, momento en el que se medirá el tiempo de inicio. Después, se introducirá el nodo restante en la topología y se volverá a esperar de nuevo a que se alcance el régimen estacionario, en ese momento se medirá el tiempo de fin.

El funcionamiento del script (apéndice F) encargado de esta medición es el siguiente:

1. Extrae la información del fichero de entrada "nodos" (cuya utilidad ya fue mencionada en la sección anterior) para poderla emplear en los procesos del script.

2. Reinicia todos los routers pertenecientes a la topología para eliminar cualquier configuración anterior

3. Procesa, uno a uno, los nodos con el fin de establecer la topología de la red, sin tener en cuenta el último nodo del fichero "nodos" que será un nodo extremo, generando por tanto una topología con diecinueve nodos.

4. Primeramente, se le asigna al nodo actual una dirección IPv6 en base a su identificador (un número de 1 a 20); a continuación, se realiza el análisis de las coordenadas X e Y de cada nodo existente en el fichero "nodos" y se comparan con las del nodo actual, en base al radio de alcance inalámbrico. Se permite conectividad directa, únicamente, con los nodos situados a un salto (visión directa), lo que se consigue mediante la utilización del comando ip6tables, estableciendo por defecto la regla de descartar todos los paquetes que no estén destinados al propio router y aceptar solamente los provenientes de los nodos a un salto.

5. Una vez terminado el establecimiento de la topología, se procede a arrancar el demonio (o demonios en el caso de OSPF-MANET) encargado de ejecutar el protocolo correspondiente a la prueba, ya sea OSPF-MANET u OLSR, en cada uno de los nodos de la red.

6. Se espera a que la topología alcance el régimen estacionario, lo que se consigue mediante la comprobación continua de la lista de nodos alcanzables de cada uno de los nodos de la red, hasta que su número de elementos coincida con el número de nodos pertenecientes a la red.

7. En cuanto se alcanza el régimen estacionario, se almacena el tiempo en segundos, que será considerado el tiempo de inicio.

8. Se configurará el nodo restante para incluirlo en la posición de la topología que debería haber ocupado desde un principio, y se espera a que se alcance el nuevo régimen estacionario que incluye a este último nodo. Una vez alcanzado se almacena el tiempo de fin. Tras realizar la diferencia entre el tiempo de fin menos el tiempo de inicio se obtiene el tiempo de convergencia de un nuevo nodo en la red.

6.2.2.5 Carga de Señalización por Nodo sin Errores

Consiste en la cantidad de tráfico de señalización generada individualmente por cada nodo, en el caso no producirse cambios ni errores en la red.
Se tomarán veinte muestras por cada nodo de la red.

Con el fin de permitir la captura de tráfico en la red inalámbrica de pruebas de los routers, se configura la interfaz inalámbrica del PC en el mismo canal de radio que los nodos y se le asigna una dirección IPv6.

El funcionamiento del script (apéndice F) encargado de esta medición es el siguiente:

1. Extrae la información del fichero de entrada "nodos" (cuya utilidad ya fue mencionada en la sección anterior) para poderla utilizar en los procesos del script.

2. Reinicia todos los routers pertenecientes a la topología para eliminar cualquier configuración anterior.

3. Procesa, uno a uno, los nodos con el fin de establecer la topología de la red, sin tener en cuenta el último nodo del fichero "nodos" que será un nodo extremo, generando por tanto una topología con diecinueve nodos.

4. Primeramente, se le asigna al nodo actual una dirección IPv6 en base a su identificador (un número de 1 a 20); a continuación, se realiza el análisis de las coordenadas X e Y de cada nodo existente en el fichero "nodos", y se comparan con las del nodo actual en base al radio de alcance inalámbrico. Se permite conectividad directa, únicamente, con los nodos situados a un salto (visión directa), lo que se consigue mediante la utilización del comando ip6tables, estableciendo por defecto la regla de descartar todos los paquetes que no estén destinados al propio router y aceptar solamente los provenientes de los nodos a un salto.

5. Una vez terminado el establecimiento de la topología, se procede a arrancar el demonio (o demonios en el caso de OSPF-MANET) encargado de ejecutar el protocolo correspondiente a la prueba, ya sea OSPF-MANET u OLSR, en cada uno de los nodos de la red.

6. Se espera a que la topología alcance el régimen estacionario, lo que se consigue mediante la comprobación continua de la lista de nodos alcanzables de cada uno

de los nodos de la red, hasta que su número de elementos coincida con el número de nodos pertenecientes a la red.

7. Se comienza la captura de tráfico de señalización del protocolo correspondiente con el programa "tshark" para cada uno de los nodos de la red (lo que se consigue indicándole al programa la dirección MAC de origen), durante un minuto.

8. Una vez finalizada, se extraen las estadísticas del tráfico mediante el programa "capinfos". Entre ellas se incluye el tráfico medio en bytes/segundo.

6.2.2.6 Carga de señalización Total sin Errores

Consiste en la cantidad total de tráfico generada por el protocolo, que viaja por la red una vez alcanzado el régimen estacionario (steady state).
Se tomarán al menos sesenta muestras.

El funcionamiento del script (apéndice F) es equivalente al anterior, con la salvedad de que en el momento de realizar la captura de tráfico no se especifica ningún filtro de nodo, con lo que el tráfico capturado será el de toda la red.

6.2.2.7 Carga de señalización Máxima Total con Caídas de Enlaces

Consiste en la carga de señalización máxima que introduce el protocolo en la red, en el momento en que se produzca la caída de un enlace entre dos nodos y se inicien todos los procesos correspondientes que involucran la creación de rutas alternativas, como ya se explicó en el capitulo dedicado al protocolo OSPF-MANET.
Utilizamos el término enlace por simplicidad; realmente, al disponerse de una única interfaz inalámbrica por nodo, no se trata de enlaces como tales, sino de conexiones punto a multipunto.
Se tomarán veinte muestras por cada uno de estos enlaces.

Con el fin de permitir la captura de tráfico en la red inalámbrica de pruebas de los routers, se configura la interfaz inalámbrica del PC en el mismo canal de radio que los nodos y se le asigna una dirección IPv6.

El funcionamiento del script (apéndice F) encargado de esta medición es el siguiente:

1. Extrae la información del fichero de entrada "nodos" (cuya utilidad ya fue mencionada en la sección anterior) para poderla emplear en los procesos del script.

2. Reinicia todos los routers pertenecientes a la topología para eliminar cualquier configuración anterior.

3. Procesa, uno a uno, los nodos con el fin de establecer la topología de la red, sin tener en cuenta el último nodo del fichero "nodos" que será un nodo extremo, generando por tanto una topología con diecinueve nodos.

4. Primeramente, se le asigna al nodo actual una dirección IPv6 en base a su identificador (un número de 1 a 20); a continuación, se realiza el análisis de las coordenadas X e Y de cada nodo existente en el fichero "nodos", y se comparan con las del nodo actual, en base al radio de alcance inalámbrico. Se permite conectividad directa, únicamente, con los nodos situados a un salto (visión directa), lo que se consigue mediante la utilización del comando ip6tables, estableciendo por defecto la regla de descartar todos los paquetes que no estén

destinados al propio router y aceptar solamente los provenientes de los nodos a un salto.

5. Una vez terminado el establecimiento de la topología, se procede a arrancar el demonio (o demonios en el caso de OSPF-MANET) encargado de ejecutar el protocolo correspondiente a la prueba, ya sea OSPF-MANET u OLSR, en cada uno de los nodos de la red.

6. Se espera a que la topología alcance el régimen estacionario, lo que se consigue mediante la comprobación continua de la lista de nodos alcanzables de cada uno de los nodos de la red, hasta que su número de elementos coincida con el número de nodos pertenecientes a la red.

7. Se comienza la captura de tráfico de señalización total de la red del protocolo correspondiente con el programa "tshark", durante un minuto.

8. Inmediatamente, se deshabilita uno de los enlaces en ambos sentidos (por ejemplo, el enlace de A a B y el enlace de B a A).

9. Una vez finalizada, se extraen las estadísticas del tráfico mediante el programa "capinfos", entre las que se incluye el tráfico medio en bytes/segundo.

10. Este proceso se repite para todos los enlaces existentes en la red.

Hay que recordar que, con el fin de realizar las pruebas de una manera coherente, estos parámetros serán medidos de la misma manera a la hora de realizar el estudio, tanto con el protocolo OSPF-MANET, como con el protocolo OLSR, adaptando los scripts únicamente para sustituir el uso de un protocolo por otro, manteniendo de esta manera la consistencia de los resultados obtenidos.

Una vez definida la batería de pruebas que será utilizada en los diferentes escenarios, sólo queda presentar los escenarios con topología aleatoria.

El número de escenarios elegido para realizar las pruebas es cinco, con la intención de representar la mayor parte de casos reales posible, dada la variedad, casi infinita, de topologías que pueden formarse en estas redes malladas inalámbricas.
Dichos escenarios fueron diseñados mediante un script creado para generar topologías aleatorias, que da como salida un fichero "nodos" con el formato correspondiente al fichero de entrada de los scripts de la batería de pruebas.
Este script escoge aleatoriamente las posiciones de cada uno de los nodos dentro de la topología, siempre cumpliendo los requisitos que se establecieron previamente y que se recuerdan a continuación:

- No ser demasiado compleja.
- Utilizar todos los nodos, en este caso veinte, y no contar con ningún nodo inaccesible a la red.
- Ofrecer la posibilidad de generar rutas alternativas en el caso de algún fallo en un nodo o enlace.

6.2.2.8 Escenario con topología aleatoria I

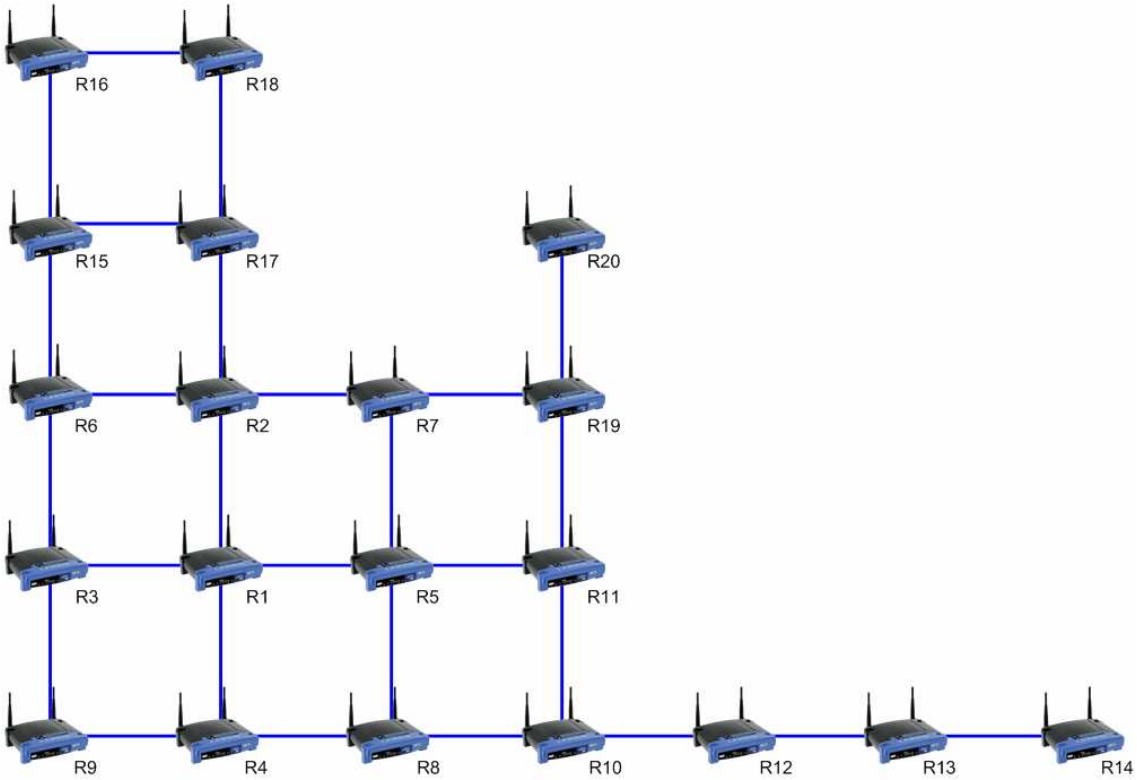


Figura 6.2 Escenario con topología aleatoria I

6.2.2.9 Escenario con topología aleatoria II

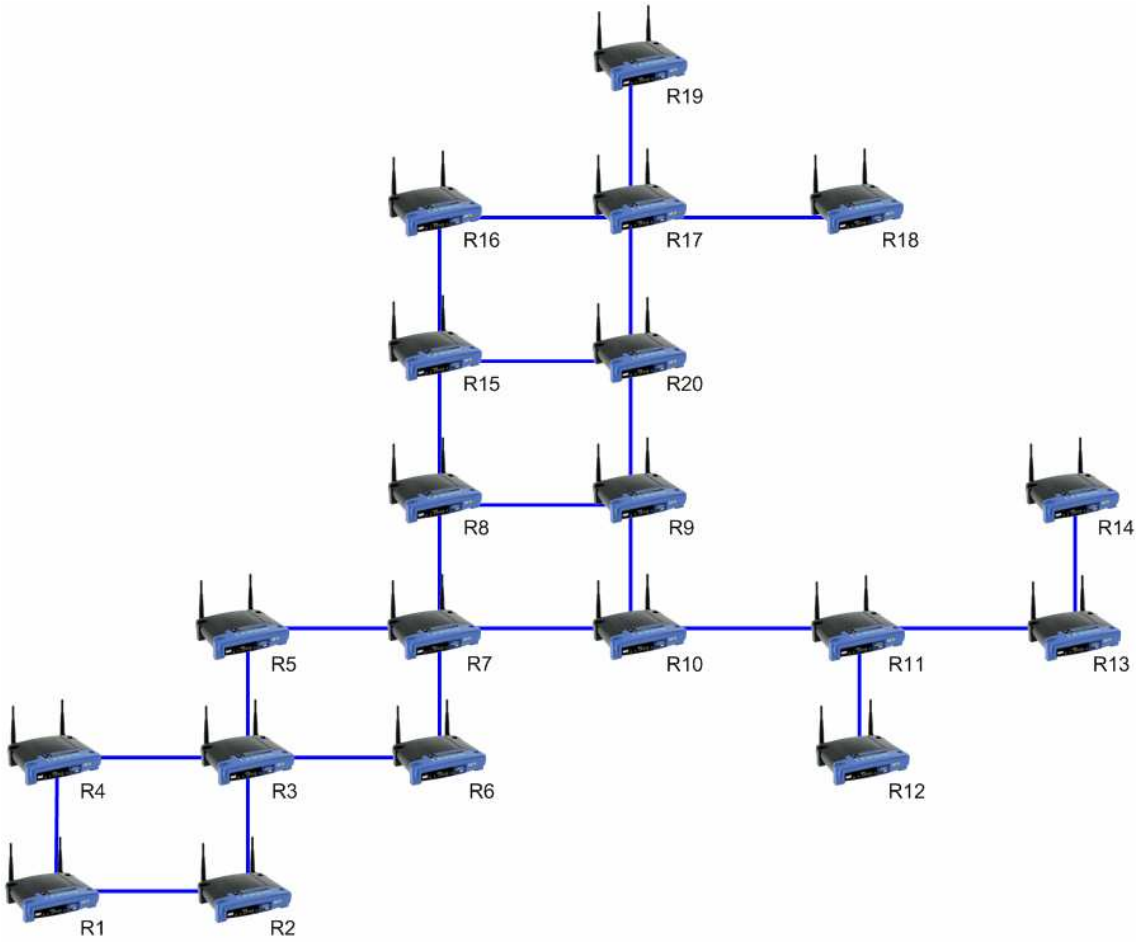


Figura 6.3 Escenario con topología aleatoria II

6.2.2.10 Escenario con topología aleatoria III

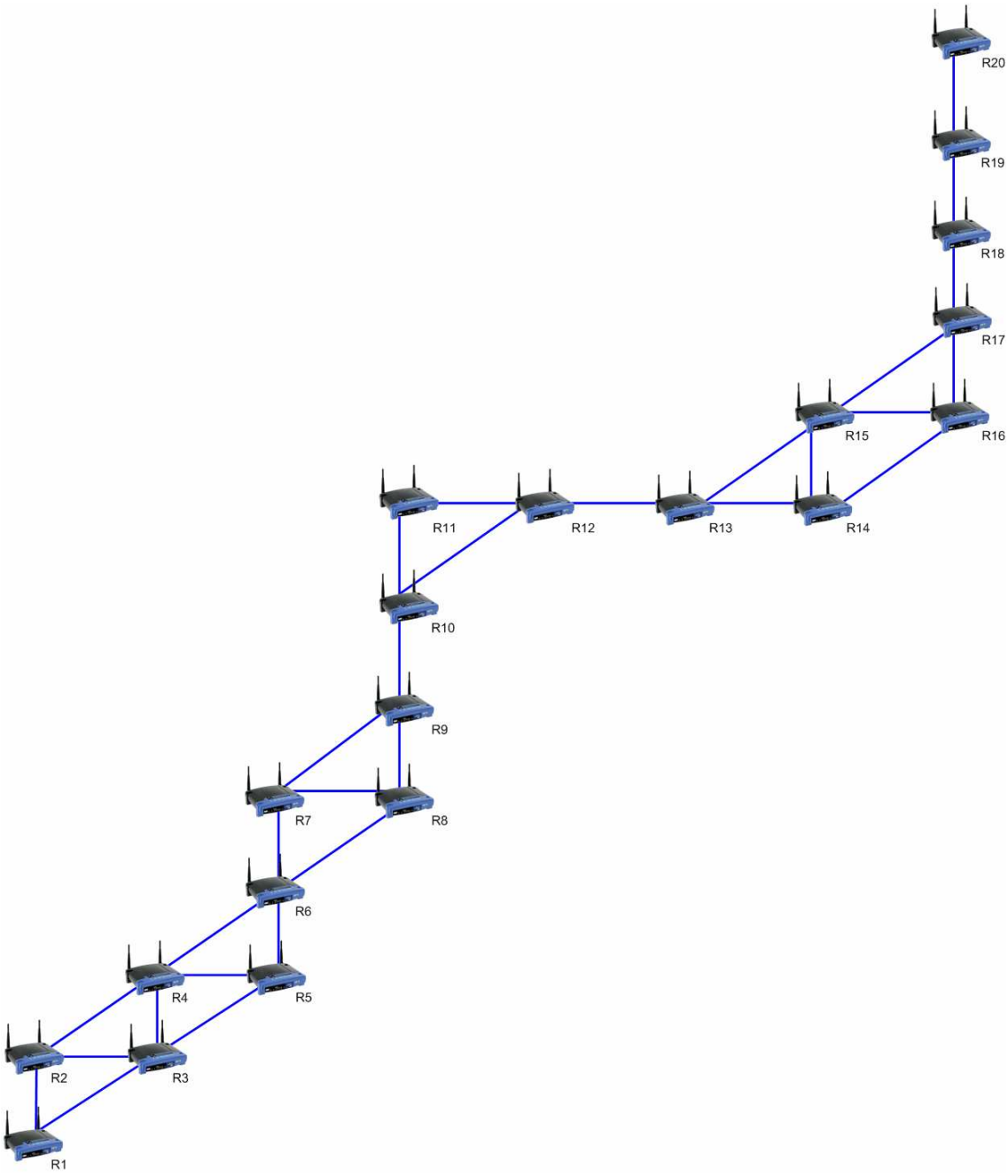


Figura 6.4 Escenario con topología aleatoria III

6.2.2.11 Escenario con topología aleatoria IV

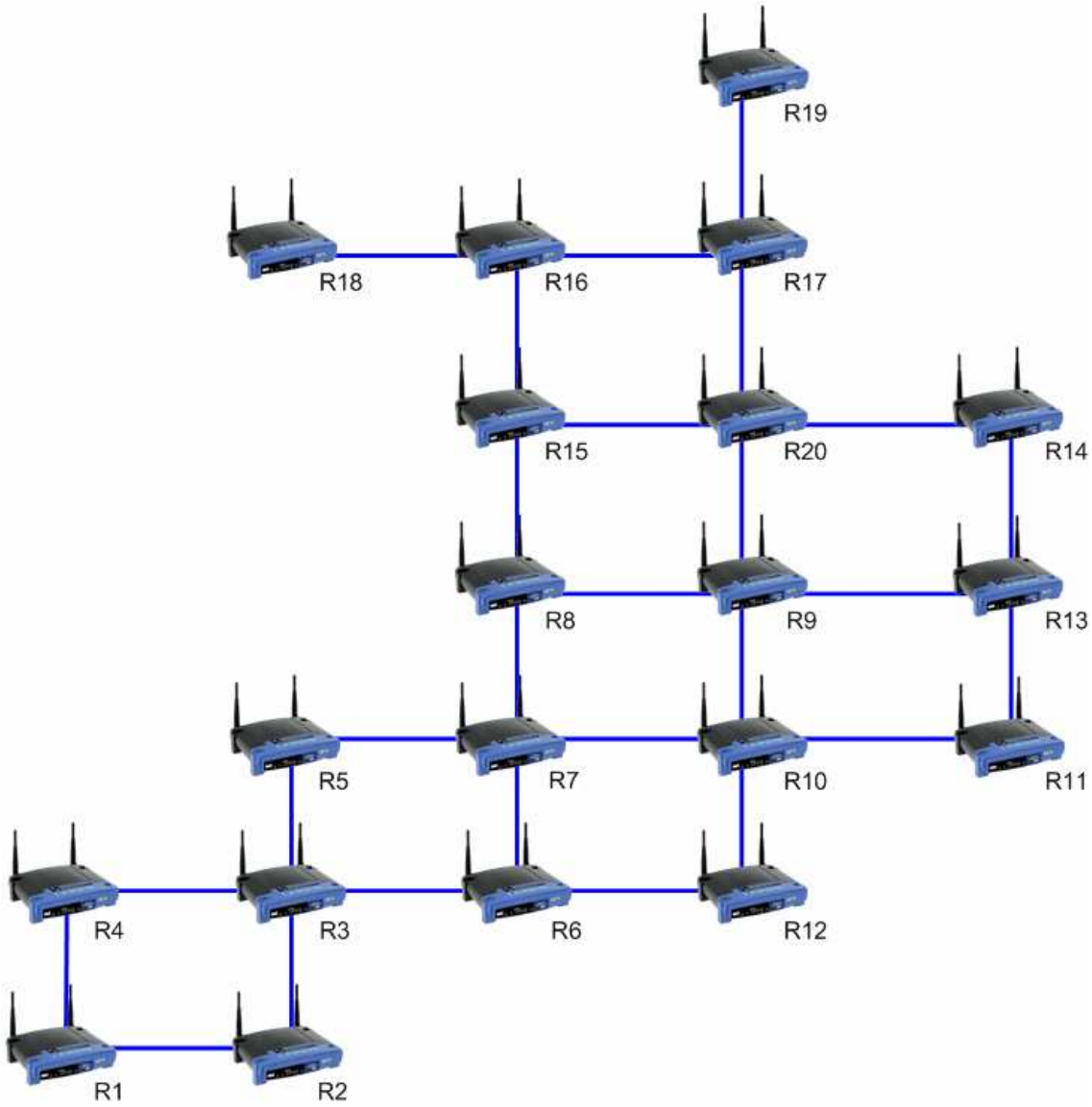


Figura 6.5 Escenario con topología aleatoria IV

6.2.2.12 Escenario con topología aleatoria V

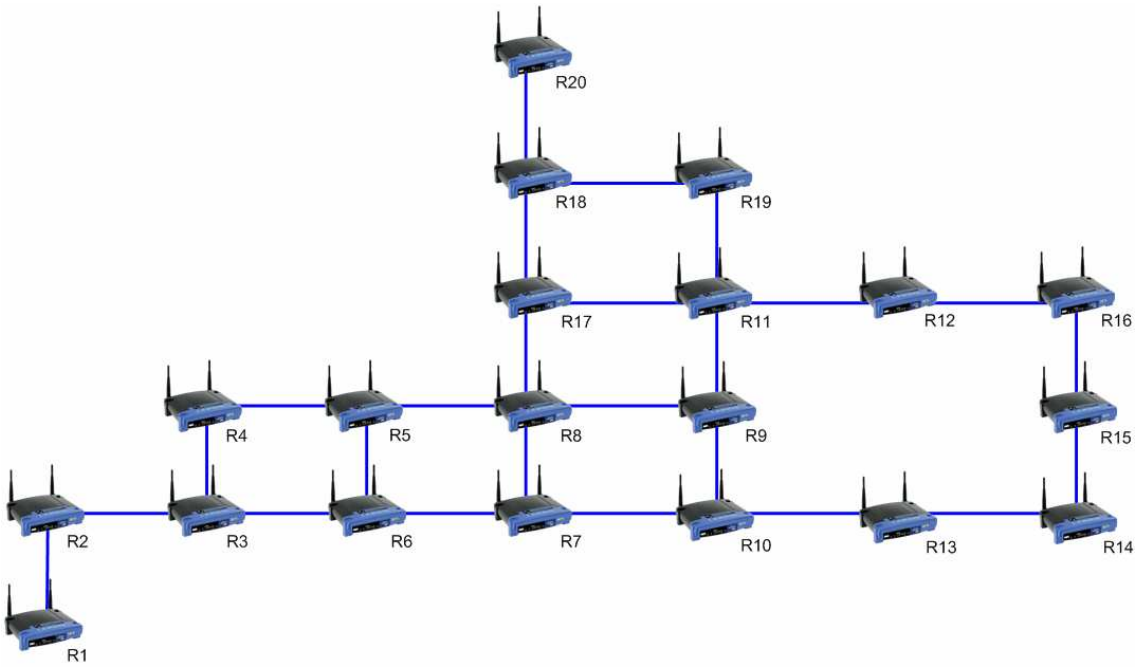


Figura 6.6 Escenario con topología aleatoria V

Capítulo 7 Resultados de las Pruebas

7.1 Introducción

Una vez analizados y descritos los protocolos OSPF-MANET y OLSR, definidos los diferentes escenarios que fueron utilizados en los experimentos, definidas las distintas pruebas que sobre los mismos se realizarían, únicamente queda presentar los resultados y conclusiones extraídas de la ejecución de la batería de pruebas sobre dichos escenarios para evaluar las prestaciones de OSPF-MANET y OLSR.

Primeramente, se presentarán los resultados del escenario con topología determinista, para OSPF-MANET y OLSR.
En segundo lugar, se presentarán los resultados de los escenarios con topología aleatoria, para OSPF-MANET y OLSR.

7.2 Resultados del Escenario con Topología Determinista

Con el fin de facilitar la interpretación de los resultados obtenidos, éstos serán representados en forma gráfica; en concreto, mediante diagramas de Caja y Bigotes (Box and Whiskers), pues son los más apropiados para visualizar el tipo de datos recogidos, proporcionando información estadística de interés, que facilitará la comparación de las diferentes medidas y escenarios.

Como referencia, ésta es una breve descripción de este tipo de diagramas:

- El análisis mediante diagrama de Caja y Bigotes resume un conjunto de observaciones de una sola variable. Proporciona una herramienta de análisis de datos exploratorios, útil para el estudio de simetría, la comprobación de hipótesis distributivas y la detección de valores atípicos. Es particularmente útil para comparar grupos de datos paralelos.

- Los datos están divididos en cuatro áreas de igual frecuencia.

- Una caja recoge el cincuenta por ciento central, donde la mediana está representada como una línea vertical dentro de la misma.

- La media estará representada por un símbolo + en color rojo.

- Las líneas horizontales, llamadas bigotes, se extienden desde cada extremo de la caja. El bigote izquierdo esta representado desde el cuartil inferior hasta dentro de 1.5 rangos intercuartílicos.

- El bigote derecho esta representado desde el cuartil superior hasta el mayor punto dentro de 1.5 rangos intercuartílicos.

- Los valores fuera de los bigotes, pero dentro de 3 rangos intercuartílicos (posibles valores atípicos), son mostrados como puntos individuales representados mediante cuadrados en color azul.

- Los puntos más distantes, confirmados como valores atípicos, se distinguen mediante un cuadrado en color azul con un símbolo + en color rojo en su interior. Los valores atípicos son puntos alejados más de 3 rangos intercuartílicos por encima o por debajo de los cuartiles superior e inferior respectivamente.

- Las barras pueden ser representadas tanto vertical como horizontalmente, en este caso lo estarán horizontalmente.

7.2.1 OSPF-MANET

Figura 7.1 Carga por nodo sin errores OSPF-MANET

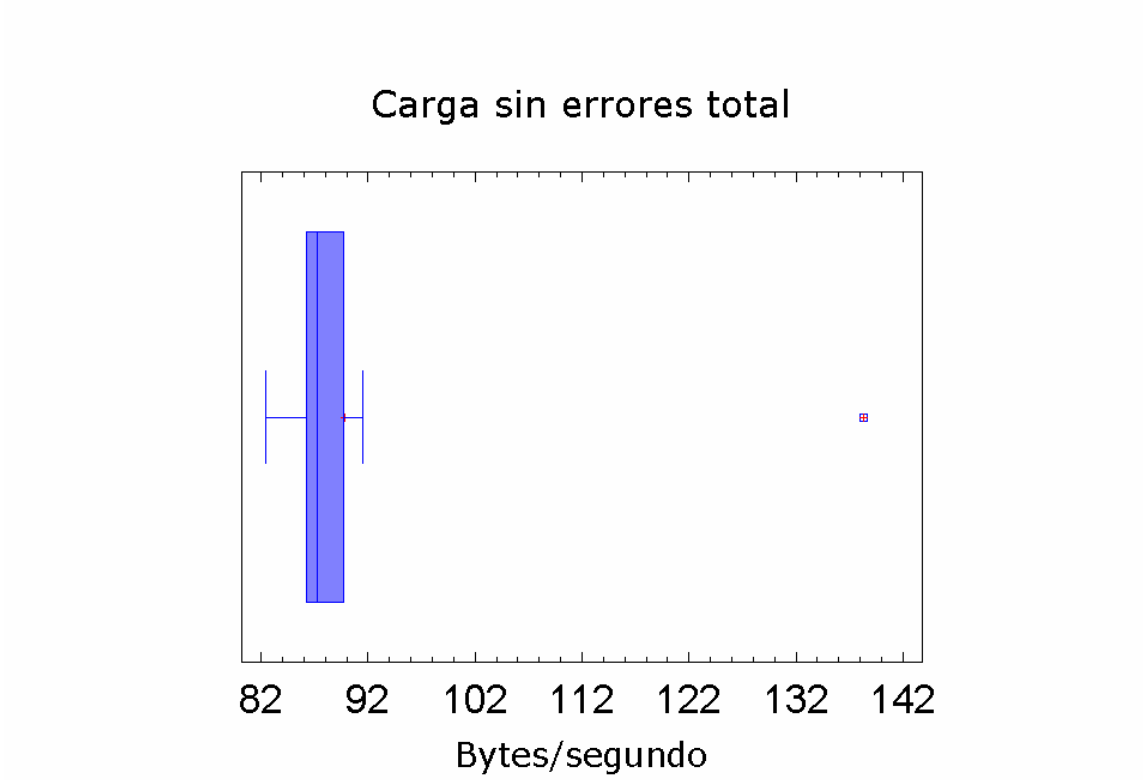


Figura 7.2 Carga total sin errores OSPF-MANET

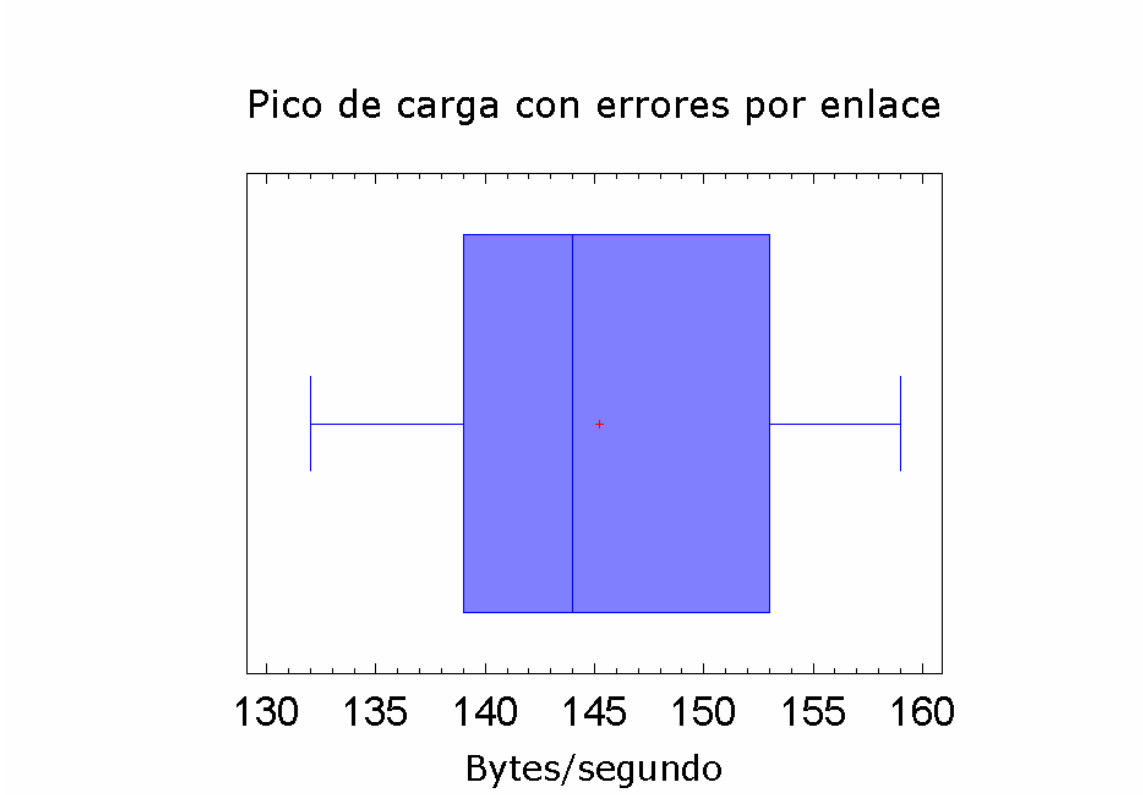


Figura 7.3 Carga máxima por enlace con errores OSPF-MANET

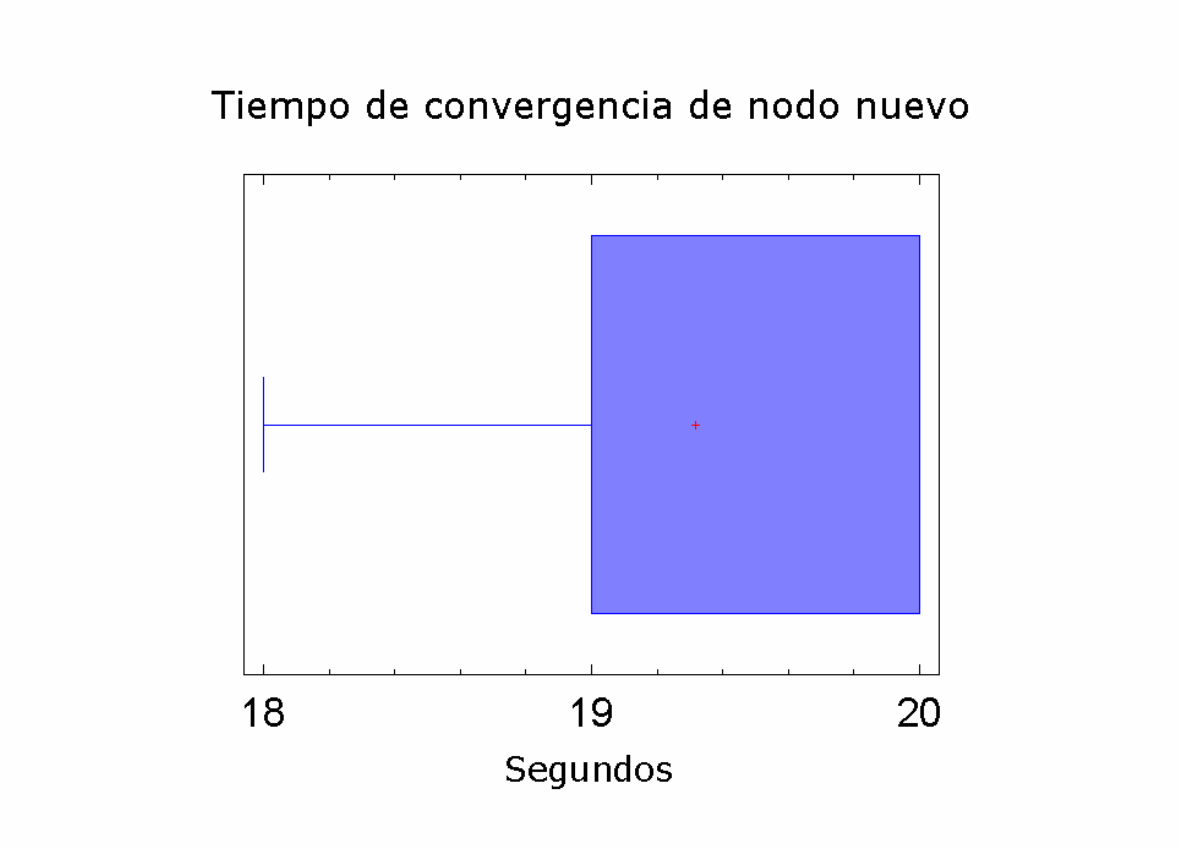


Figura 7.4 Tiempo de convergencia de nodo nuevo OSPF-MANET

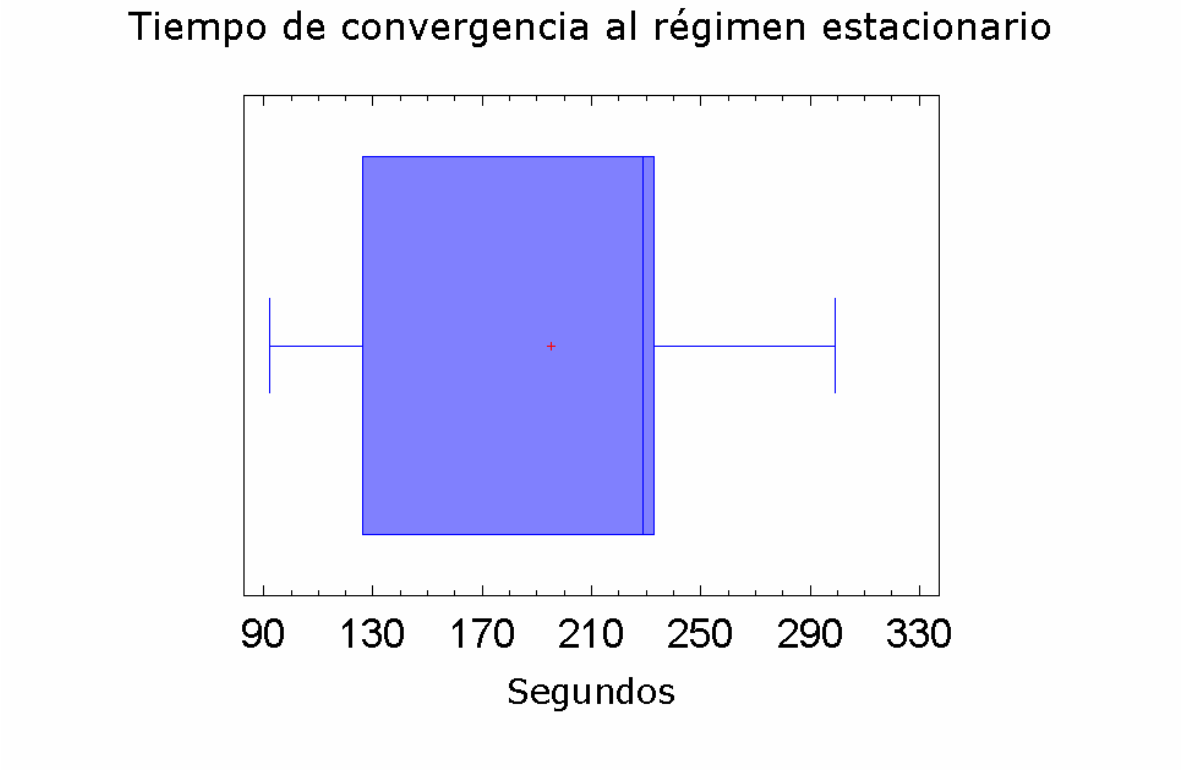


Figura 7.5 Tiempo de convergencia al régimen estacionario OSPF-MANET

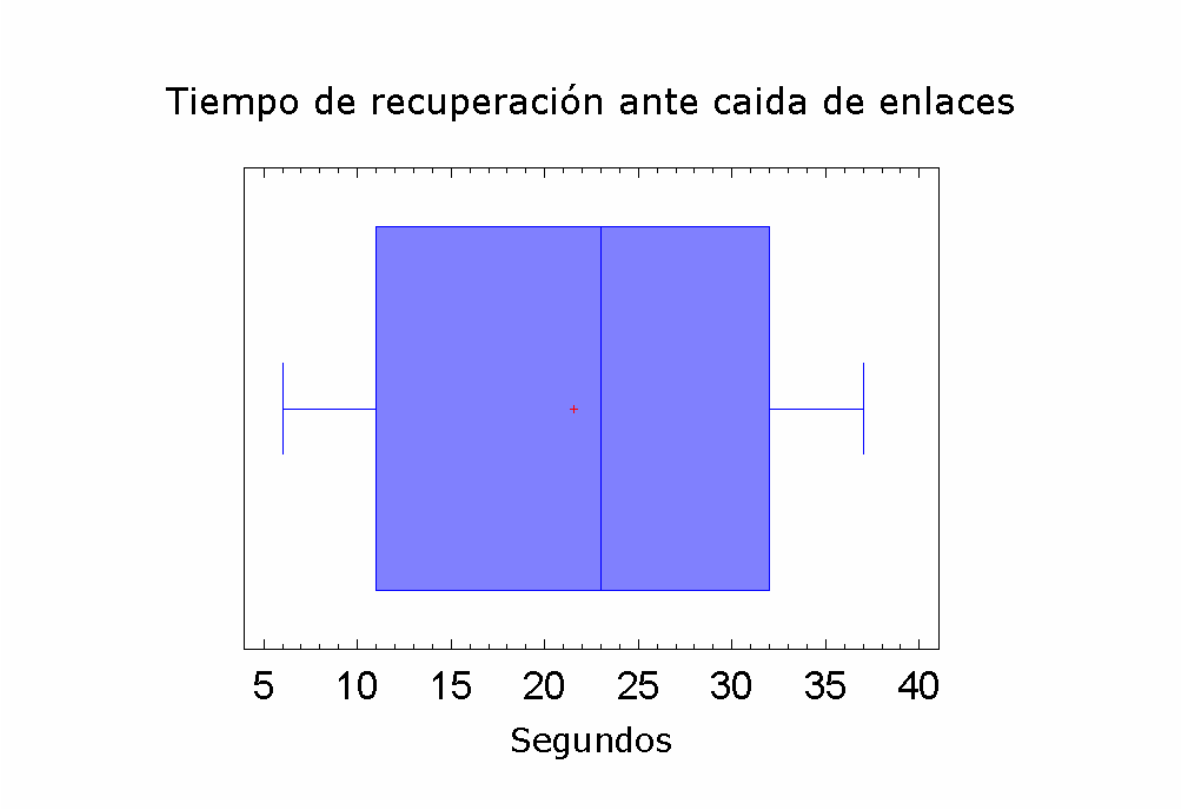


Figura 7.6 Tiempo de recuperación ante caída de enlaces OSPF-MANET

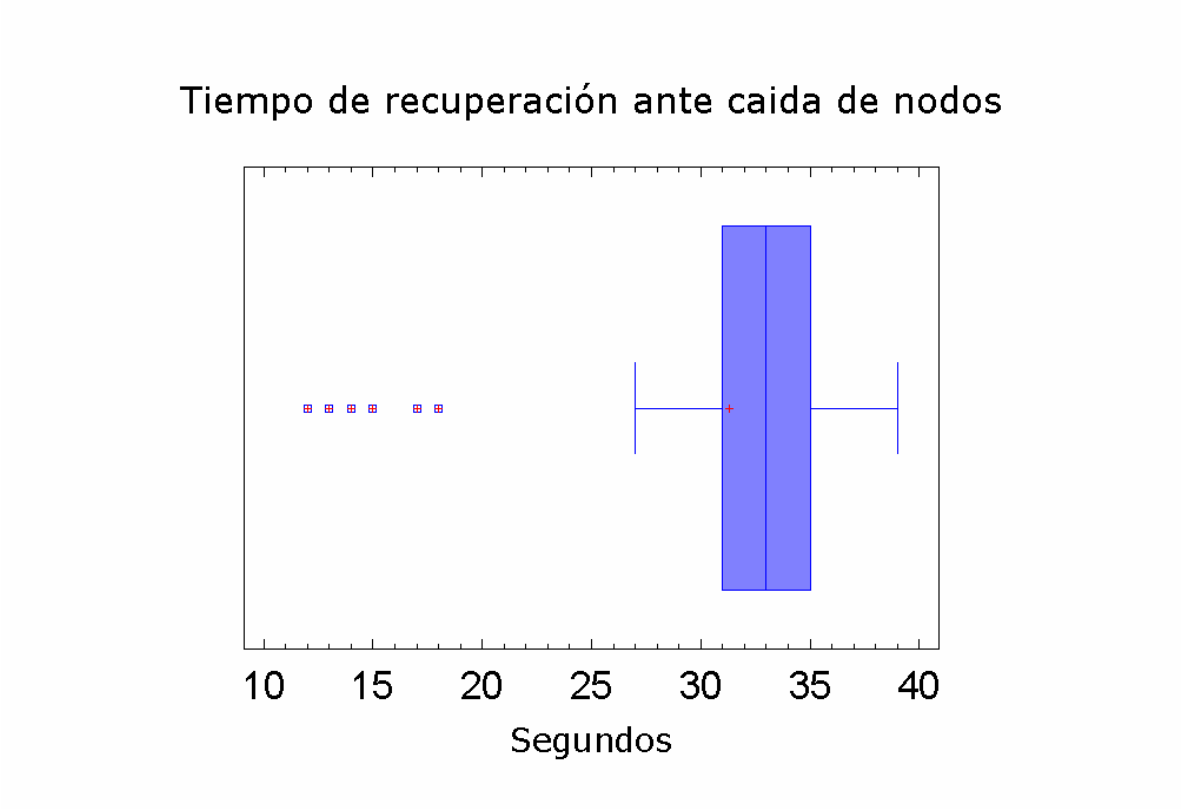


Figura 7.7 Tiempo de recuperación ante caída de nodos OSPF-MANET

7.2.1.1 Primeras Conclusiones para OSPF-MANET

Los primeros resultados en la utilización de este protocolo parecen muy alentadores.
En situaciones en las que no hay cambios o errores en la red (figuras 7.1 y 7.2), OSPF-MANET enviará únicamente una pequeña cantidad de tráfico de control; ello produce una carga mínima en la red, teniendo unos valores promedio de 12 Bytes/segundo por nodo en estado estable, lo que es un valor excelente, y valores en torno a 89 Bytes/segundo en toda la red (un valor proporcionado, dado que la componen nueve nodos).

La respuesta del protocolo a la hora de producirse cambios o fallos en la red (figura 7.3) también es muy buena, generándose, eso sí, una mayor cantidad de tráfico de control, comparándose con el caso en el que no se producen errores; esta es debida al funcionamiento especifico de OSPF-MANET que, al detectar un cambio, comienza a generar mensajes para que todos los nodos actualicen sus tablas de encaminamiento.
Al deshabilitar (simulando caída) cada uno de los enlaces entre dos nodos, se obtienen valores promedio de carga entre 132 y 159 Bytes/segundo en la totalidad de red.
Al contrario que en las redes cableadas, donde la mayor parte del tráfico es absorbido por el backbone, las redes malladas inalámbricas tienen la ventaja de que el tráfico total de señalización está repartido equilibradamente entre los diferentes enlaces, evitando saturar una única ruta, lo que permitiría absorber grandes cantidades de trafico, si fuera necesario.

En cuanto a los tiempos medidos, los resultados ya no son tan buenos para el número de nodos que se están empleando, lo que era de esperar tratándose de un protocolo proactivo, aunque, eso sí, no llegan a ser malos.
Tiempos de 18 segundos para introducir un nuevo nodo a la red (figura 7.4) y 195 segundos desde que se conectan los routers hasta que toda la red alcanza un estado estable (figura 7.5) son resultados aceptables, aunque mejorables.

Los tiempos de recuperación de rutas, a la hora de producirse caídas de nodos (figura 7.7), están en torno a 31 segundos, un resultado algo elevado que requerirá estar muy atentos a la hora de ejecutar las pruebas en las topologías aleatorias, para ver su comportamiento en el momento en el que el número de nodos pertenecientes a la red sea elevado.

El tiempo de recuperación de rutas al producirse fallos en enlaces (figura 7.6) es inferior, 21 segundos, un resultado aceptable que habrá que controlar, también, a la hora de escalar la red.

Todos estos tiempos son, como cabía esperar y ya se ha mencionado, algo elevados motivados, de nuevo, por el funcionamiento de OSPF-MANET a la hora de producirse cambios en la red. Debido a que OSPF-MANET comienza el proceso de creación de nuevas rutas al detectar un cambio, los tiempos de respuesta se ven afectados negativamente, pues al tiempo básico se le sumará el tiempo que tardan en establecerse y propagarse las nuevas rutas.

Se ha detectado un problema de estabilidad en el funcionamiento del demonio ospf6d que, aleatoriamente, deja de ejecutarse, teniendo que volver a arrancarlo manualmente para que entre de nuevo en funcionamiento. La investigación realizada apunta a que, con otra implementación del mismo, éste problema no se produciría, pero el objeto de este proyecto es únicamente la implementación que se está usando, por lo que no desviaremos la atención de este fin.

Por tanto, nos queda observar cómo se comportará el protocolo a la hora de escalar la red con un número elevado de nodos, comprobación que se realizará con las topologías aleatorias. Teniendo en cuenta el número de nodos actual y la topología establecida, los resultados obtenidos en este primer experimento son bastante prometedores.

7.2.2 OLSR

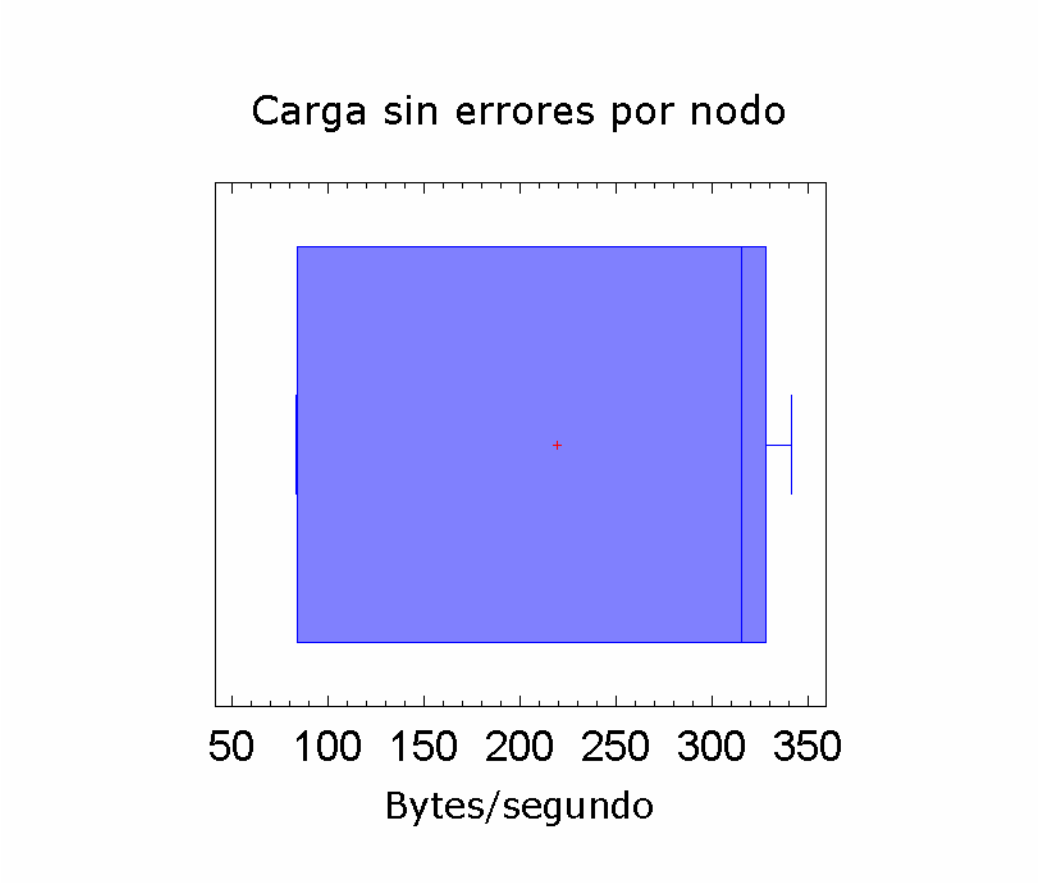


Figura 7.8 Carga por nodo sin errores OLSR

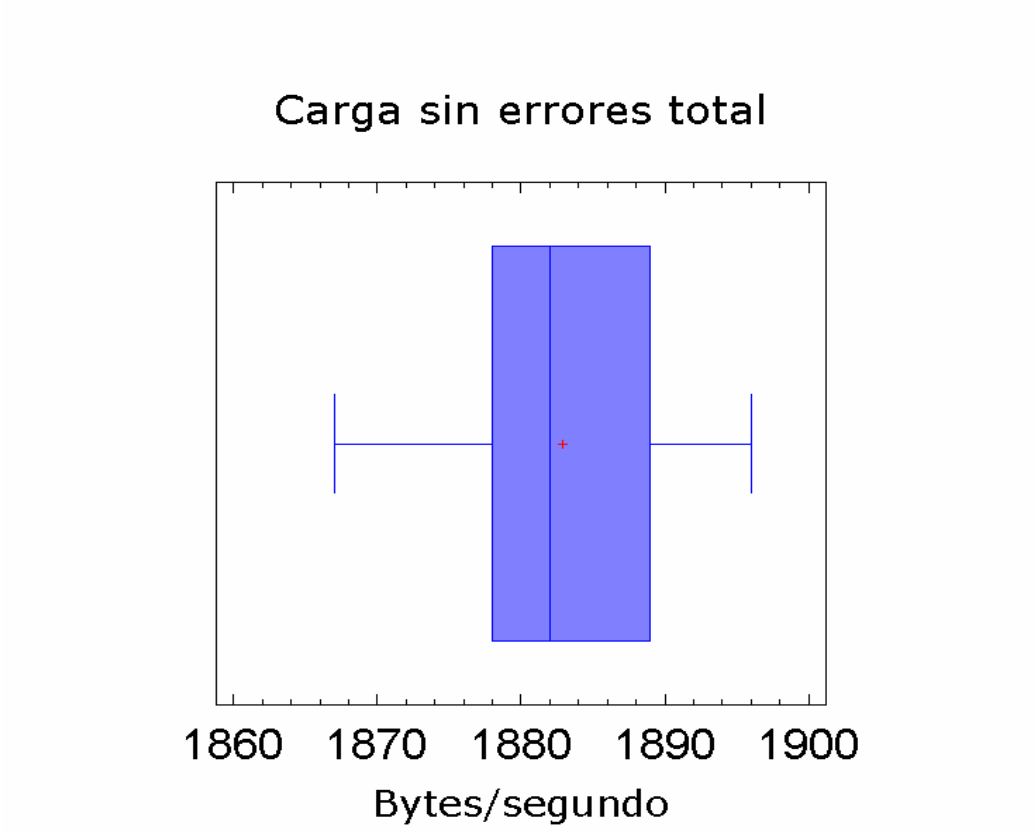


Figura 7.9 Carga total sin errores OLSR

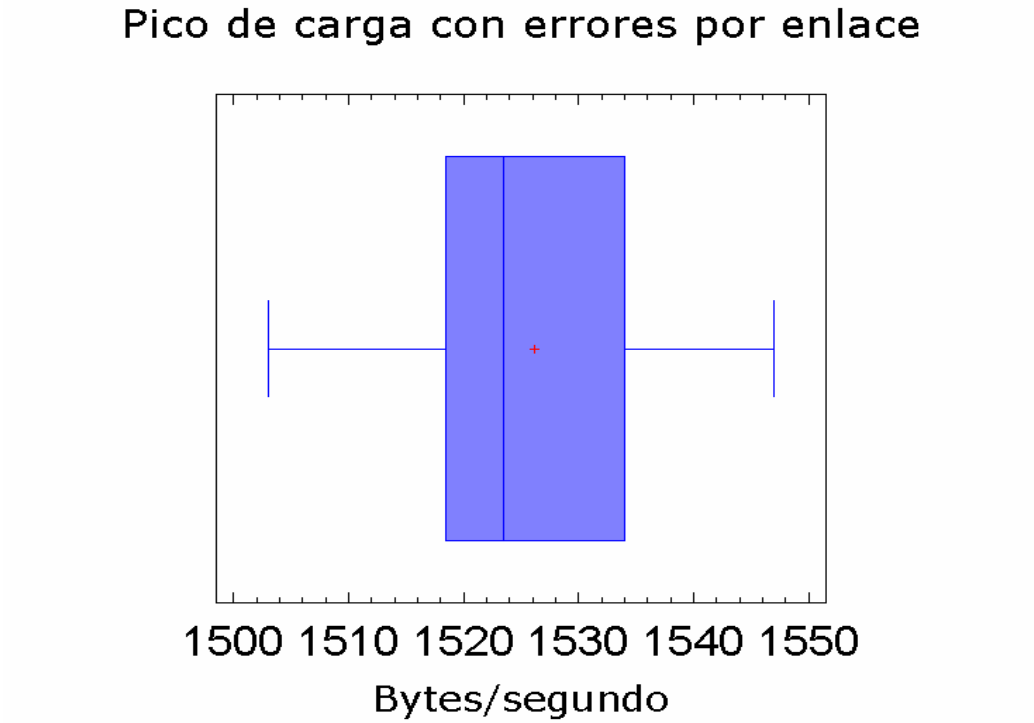


Figura 7.10 Carga máxima por enlace con errores OLSR

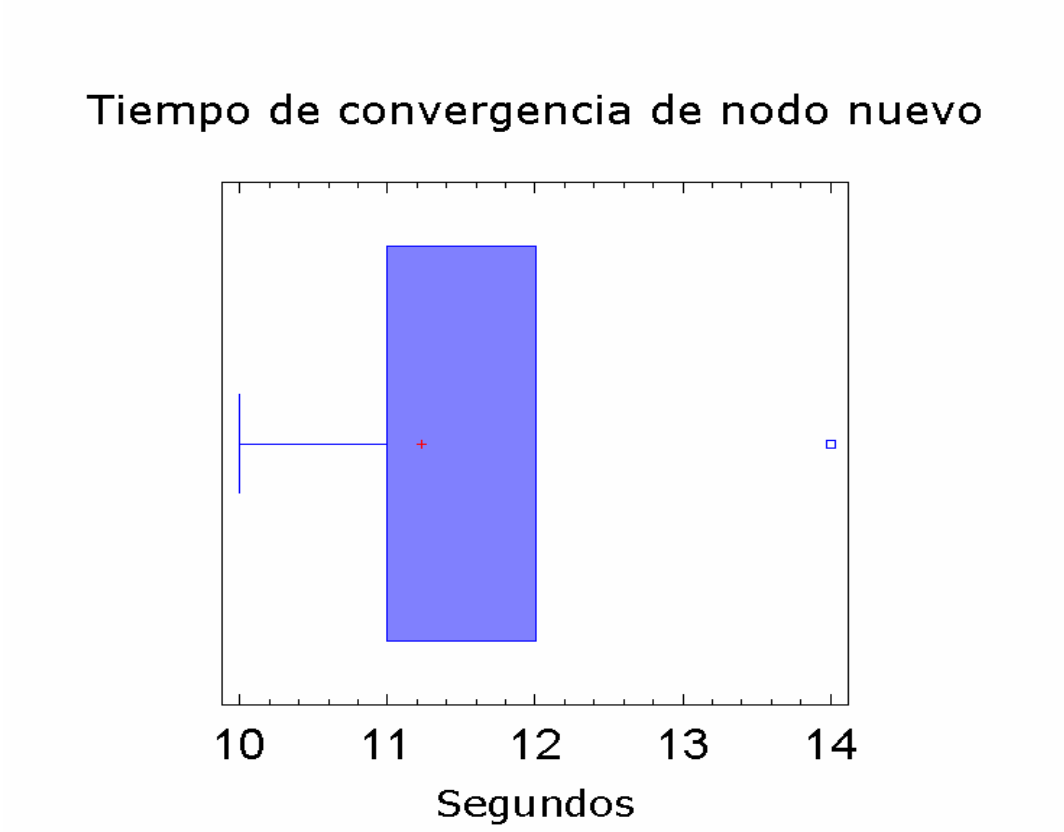


Figura 7.11 Tiempo de convergencia de nodo nuevo OLSR

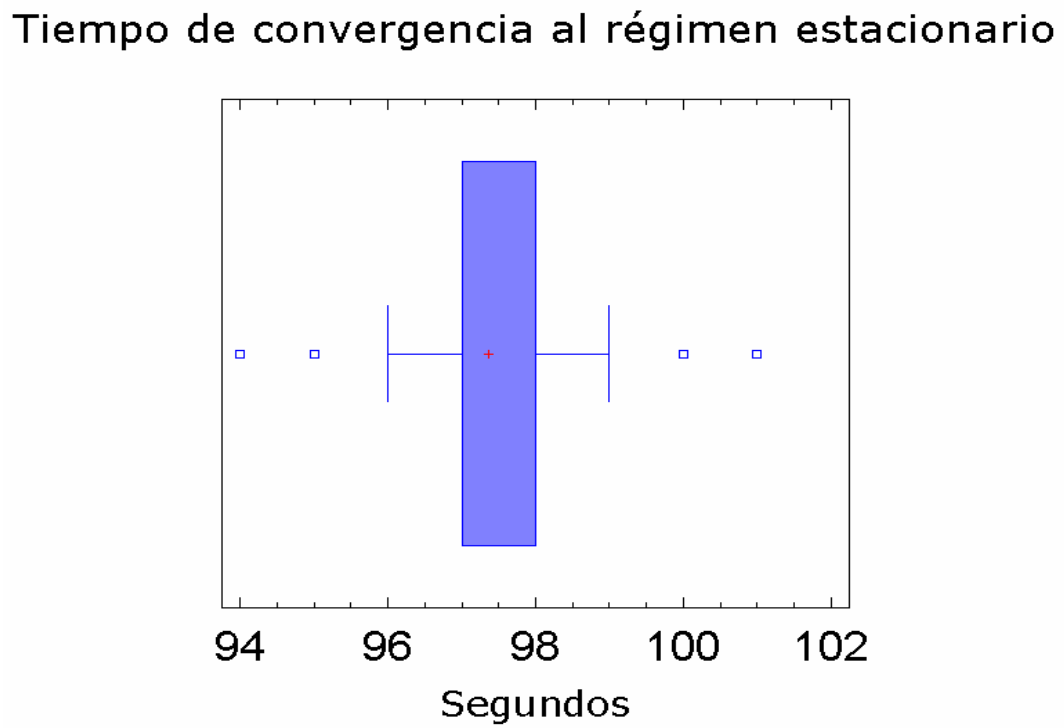


Figura 7.12 Tiempo de convergencia al régimen estacionario OLSR

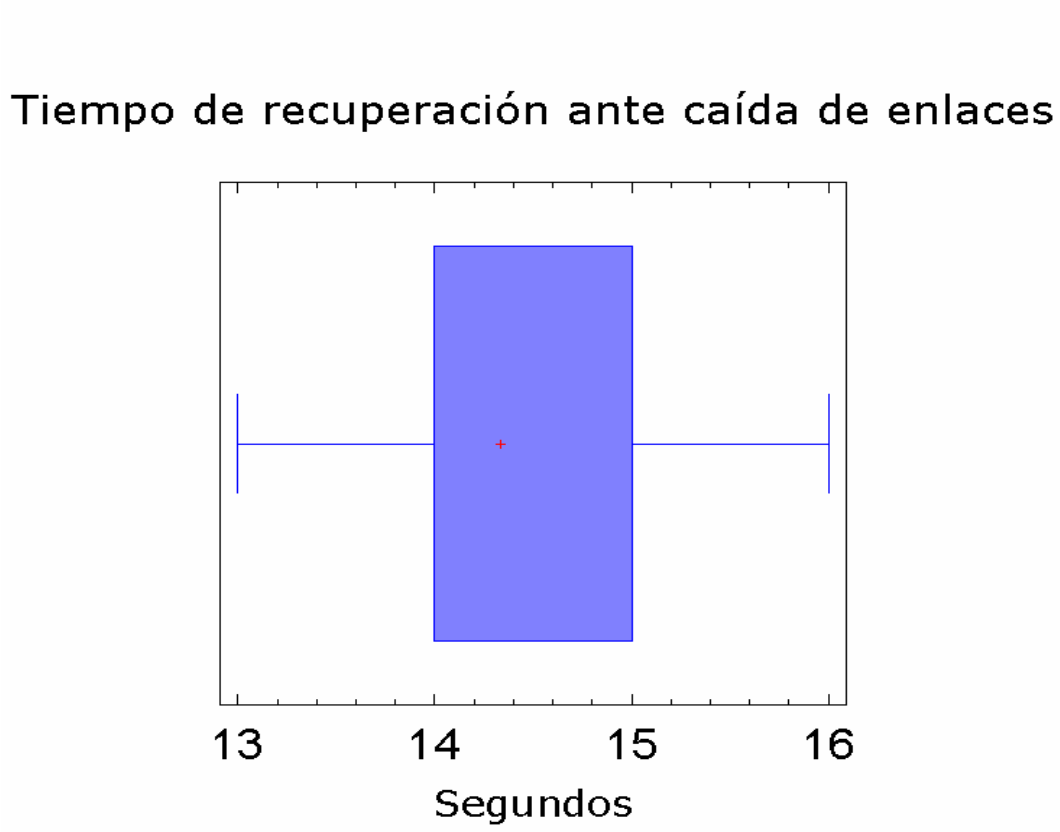


Figura 7.13 Tiempo de recuperación ante caída de enlaces OLSR

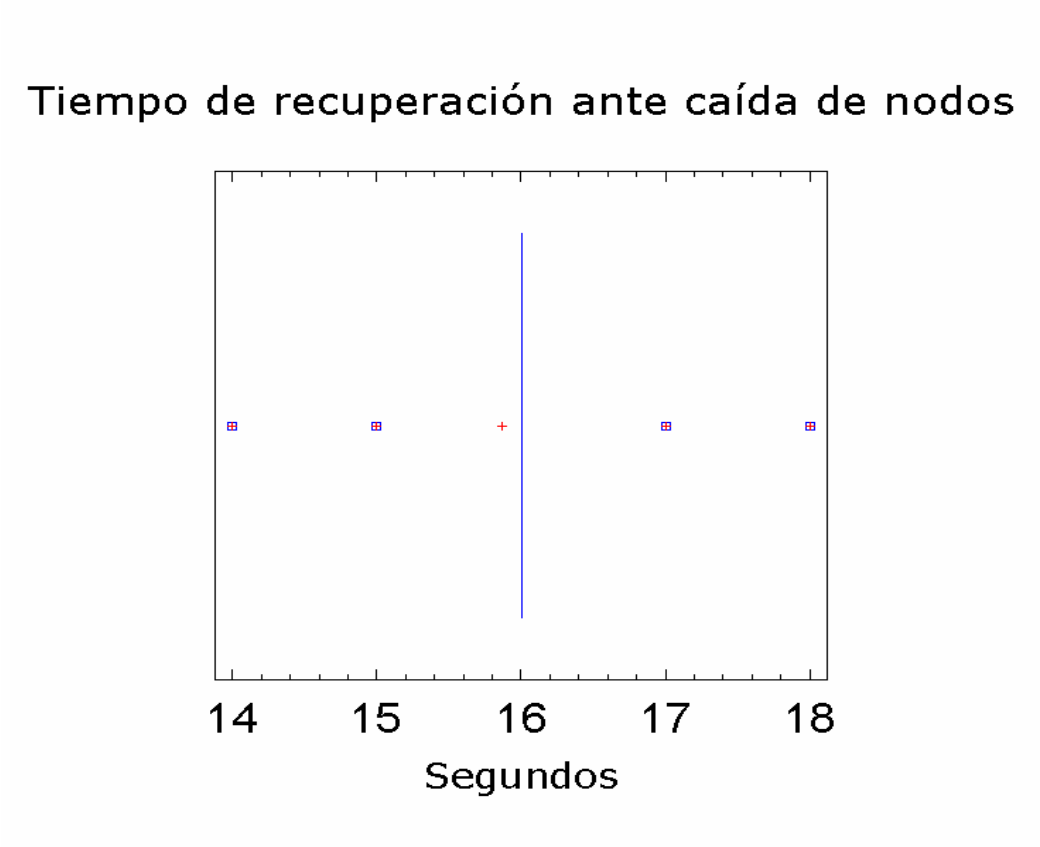


Figura 7.14 Tiempo de recuperación ante caída de nodos OLSR

7.2.2.1 Primeras Conclusiones para OLSR

Una vez obtenidos los primeros resultados para OLSR es posible comparar el comportamiento de ambos protocolos, OSPF-MANET y OLSR.

En cuanto a la carga introducida en la red (figuras 7.8 y 7.9), se puede apreciar un aumento general en el tráfico de control generado, para todas las medidas. Este comportamiento es causado por el funcionamiento de OLSR, ya que requiere el intercambio de una mayor cantidad de tráfico de control y con cadencias más frecuentes (a intervalos de tiempo más reducidos), en comparación con OSPF-MANET.

La carga de señalización introducida por cada nodo cuando no se producen cambios en la red tiene un valor promedio de 330 bytes/segundo. En la figura 7.8, pueden observarse ciertos valores puntuales mucho menores, en torno a 84 bytes/segundo. El motivo de esta diferencia, es el resultado de la utilización del mecanismo de inundación por MPRs.
En este caso, los valores en torno a 330 bytes/segundo corresponden a los nodos seleccionados como MPRs, encargados de distribuir el tráfico del resto de nodos, mientras que los valores en torno a 84 bytes/segundo corresponden a los nodos que no han sido elegidos como tales y que reenviarán sus mensajes únicamente a su nodo MPR correspondiente.
En cuanto a la carga total introducida cuando no se producen cambios en la red (figura 7.9) se obtiene un valor promedio en torno a 1880 bytes/segundo.

La carga de señalización introducida cuando se producen cambios en la red (caída de un enlace entre dos nodos, figura 7.10) es similar al caso en el que no se producían errores; esto es debido a que, al contrario que OSPF-MANET, OLSR simplemente eliminará la entrada correspondiente en sus tablas, cuando, tras varios intercambios de mensajes de control, ésta no se haya actualizado.
Concretamente, los valores promedio obtenidos son en torno a 1530 bytes/segundo.

OLSR muestra una mejora global en los tiempos de respuesta, ya que dispone en todo momento de rutas alternativas almacenadas en sus tablas; ello permite que, al producirse un cambio, no haya que esperar a que se generen nuevas rutas alterativas (como ocurría en OSPF-MANET), lo que implica una reducción de este tiempo.
En comparación con OSPF-MANET, los tiempos de respuesta se reducen aproximadamente a la mitad. Concretamente, el tiempo de convergencia de un nodo nuevo (figura 7.11) es, en promedio, de 11 segundos, mientras que el tiempo de convergencia total (figura 7.12) promedio es de 97 segundos.

En cuanto a los tiempos de recuperación ante caída de enlaces (figura 7.13) y nodos (figura 7.14) son, en promedio, en torno a 14 y 16 segundos, respectivamente.

7.3 Resultados de los Escenarios con Topología Aleatoria

Dado que la cantidad de medidas realizadas en estos escenarios es muy elevada, su representación mediante gráficas sería excesivamente extensa, dificultando la visión de conjunto y las posibilidades de comparación.
Por ello, se ha preferido su representación mediante tablas.

OSPF-MANET 20 Nodos Escenario I						
Cargas en Bytes/segundo Tiempos en Segundos	Mínimo	Media	Máximo	Mediana	Desviación Típica	Intervalos de Confianza (95%)
Carga Sin Errores por Nodo	11	11,6	12	12	0,5	+/- 0,2
Carga Sin Errores Total	188,6	195,9	200,1	196,9	3,7	+/- 0,9
Pico de Carga con Errores de Enlace	161	310,8	336	317	19,8	+/- 7,8
Tiempo de Convergencia de Nodo Nuevo	36	37,6	38	38	0,6	+/- 0,2
Tiempo de Convergencia Total	199	205,2	212	203	0,8	+/- 0,4
Tiempo de Recuperación ante Caída de Enlaces	10	19,4	27	19	4,3	+/- 1,3
Tiempo de Recuperación ante Caída de Nodos	12	29,4	36	35	8,4	+/- 2,6

Tabla 7.1OSPF-MANET Escenario aleatorio I

OSPF-MANET 20 Nodos Escenario II						
Cargas en Bytes/segundo Tiempos en Segundos	Mínimo	Media	Máximo	Mediana	Desviación Típica	Intervalos de Confianza (95%)
Carga Sin Errores por Nodo	10	11,6	13	12	0,6	+/- 0,3
Carga Sin Errores Total	188,6	195,9	201,5	196,9	3,7	+/- 0,9
Pico de Carga con Errores de Enlace	258	293,5	321	300	16,5	+/- 6,9
Tiempo de Convergencia de Nodo Nuevo	36	37,1	39	37	0,5	+/- 0,2
Tiempo de Convergencia Total	195	195,4	197	195	0,7	+/- 0,3
Tiempo de Recuperación ante Caída de Enlaces	12	20,8	26	22	3,3	+/- 1
Tiempo de Recuperación ante Caída de Nodos	12	29,2	37	36	9,3	+/- 2,6

Tabla 7.2 OSPF-MANET Escenario aleatorio II

OSPF-MANET 20 Nodos Escenario III						
Cargas en Bytes/segundo Tiempos en Segundos	Mínimo	Media	Máximo	Mediana	Desviación Típica	Intervalos de Confianza (95%)
Carga Sin Errores por Nodo	11	11,9	14	12	0,7	+/- 0,3
Carga Sin Errores Total	181,7	198,3	241,5	198,6	6,8	+/- 1,7
Pico de Carga con Errores de Enlace	274	303,8	324	304	13,2	+/- 5
Tiempo de Convergencia de Nodo Nuevo	35	37,3	38	37	0,6	+/- 0,2
Tiempo de Convergencia Total	206	207,4	208	208	0,6	+/- 0,2
Tiempo de Recuperación ante Caída de Enlaces	16	21,8	26	22	3	+/- 0,8
Tiempo de Recuperación ante Caída de Nodos	12	29,6	37	33	8,3	+/- 2,5

Tabla 7.3 OSPF-MANET Escenario aleatorio III

OSPF-MANET 20 Nodos Escenario IV						
Cargas en Bytes/segundo Tiempos en Segundos	Mínimo	Media	Máximo	Mediana	Desviación Típica	Intervalos de Confianza (95%)
Carga Sin Errores por Nodo	11	11,6	14	12	0,7	+/- 0,2
Carga Sin Errores Total	181,7	196,5	204,9	197,8	5,2	+/- 1,3
Pico de Carga con Errores de Enlace	267	307,3	329	309	14,5	+/- 5,7
Tiempo de Convergencia de Nodo Nuevo	36	37,7	38	38	0,5	+/- 0,2
Tiempo de Convergencia Total	205	207,5	208	208	0,8	+/- 0,2
Tiempo de Recuperación ante Caída de Enlaces	17	22,1	25	22	2,4	+/- 0,7
Tiempo de Recuperación ante Caída de Nodos	12	32,7	37	37	7,2	+/- 2

Tabla 7.4 OSPF-MANET Escenario aleatorio IV

OSPF-MANET 20 Nodos Escenario V						
Cargas en Bytes/segundo Tiempos en Segundos	Mínimo	Media	Máximo	Mediana	Desviación Típica	Intervalos de Confianza (95%)
Carga Sin Errores por Nodo	10	11,6	15	11	1,1	+/- 0,5
Carga Sin Errores Total	173,4	194,5	204,9	196,3	6,9	+/- 1,8
Pico de Carga con Errores de Enlace	263	299,3	320	300	13,4	+/- 5,5
Tiempo de Convergencia de Nodo Nuevo	35	37,4	38	38	0,8	+/- 0,2
Tiempo de Convergencia Total	203	207,2	208	208	1,1	+/- 0,3
Tiempo de Recuperación ante Caída de Enlaces	17	23,6	26	25	2,2	+/- 0,6
Tiempo de Recuperación ante Caída de Nodos	12	30	37	35	8,9	+/- 2,7

Tabla 7.5 OSPF-MANET Escenario aleatorio V

OLSR 20 Nodos Escenario I						
Cargas en Bytes/segundo Tiempos en Segundos	Mínimo	Media	Máximo	Mediana	Desviación Típica	Intervalos de Confianza (95%)
Carga Sin Errores por Nodo	83	308,7	388	335	80,2	+/- 37,5
Carga Sin Errores Total	5868,5	6025,8	6159,2	6029	72,12	+/- 18,9
Pico de Carga con Errores de Enlace	4418,8	5886,2	6125	5916,3	166,8	+/- 14,1
Tiempo de Convergencia de Nodo Nuevo	23	23,3	25	23	0,7	+/- 0,2
Tiempo de Convergencia Total	152	153,5	160	153	1,6	+/- 0,75
Tiempo de Recuperación ante Caída de Enlaces	10	16	34	13,5	7,9	+/- 3,7
Tiempo de Recuperación ante Caída de Nodos	11	22,6	33	23,5	6,5	+/- 2,1

Tabla 7.6 OLSR Escenario aleatorio I

OLSR 20 Nodos Escenario II						
Cargas en Bytes/segundo Tiempos en Segundos	**Mínimo**	**Media**	**Máximo**	**Mediana**	**Desviación Típica**	**Intervalos de Confianza (95%)**
Carga Sin Errores por Nodo	82	264,1	352	303	97,7	+/- 47
Carga Sin Errores Total	5062,1	5244,5	5391,4	5249,9	71,5	+/- 18,8
Pico de Carga con Errores de Enlace	4301,3	5094,5	5335,6	5128,9	149,7	+/- 13,4
Tiempo de Convergencia de Nodo Nuevo	15	21,7	25	22,5	3,2	+/- 1,5
Tiempo de Convergencia Total	146	149,4	154	151	3,1	+/- 1,4
Tiempo de Recuperación ante Caída de Enlaces	8	14	26	15	8,9	+/- 3,6
Tiempo de Recuperación ante Caída de Nodos	7	16,9	32	14	7,8	+/- 2,5

Tabla 7.7 OLSR Escenario aleatorio II

OLSR 20 Nodos Escenario III						
Cargas en Bytes/segundo Tiempos en Segundos	**Mínimo**	**Media**	**Máximo**	**Mediana**	**Desviación Típica**	**Intervalos de Confianza (95%)**
Carga Sin Errores por Nodo	77,1	278,1	379,4	301,3	85,4	+/- 8,4
Carga Sin Errores Total	4738,8	5523,4	5768,8	5563,9	169,6	+/- 44,5
Pico de Carga con Errores de Enlace	4086,4	5340,7	5842,9	5418,6	314,1	+/- 25,6
Tiempo de Convergencia de Nodo Nuevo	22	23,5	34	23	2,3	+/- 0,7
Tiempo de Convergencia Total	154	160	162	161	1,6	+/- 0,7
Tiempo de Recuperación ante Caída de Enlaces	10	16,8	34	14	7,2	+/- 2,1
Tiempo de Recuperación ante Caída de Nodos	7	22,2	33	23	7	+/- 2,2

Tabla 7.8 OLSR Escenario aleatorio III

OLSR 20 Nodos Escenario IV						
Cargas en Bytes/segundo Tiempos en Segundos	**Mínimo**	**Media**	**Máximo**	**Mediana**	**Desviación Típica**	**Intervalos de Confianza (95%)**
Carga Sin Errores por Nodo	78,4	313,1	400,2	334,7	80,7	+/- 7,9
Carga Sin Errores Total	5967	6119,4	6342,6	6108,1	70,6	+/- 18,5
Pico de Carga con Errores de Enlace	4294,4	6011,8	6310,2	6080,4	289,3	+/- 24,5
Tiempo de Convergencia de Nodo Nuevo	16	22,8	25	23	1,33	+/- 0,4
Tiempo de Convergencia Total	153	154,7	160	154,5	1,6	+/- 0,6
Tiempo de Recuperación ante Caída de Enlaces	10	15,4	34	12	6,5	+/- 2
Tiempo de Recuperación ante Caída de Nodos	9	16,5	30	14	6,1	+/- 1,7

Tabla 7.9 OLSR Escenario aleatorio IV

OLSR 20 Nodos Escenario V						
Cargas en Bytes/segundo Tiempos en Segundos	**Mínimo**	**Media**	**Máximo**	**Mediana**	**Desviación Típica**	**Intervalos de Confianza (95%)**
Carga Sin Errores por Nodo	81	297,7	376,9	316,7	74,7	+/- 7,3
Carga Sin Errores Total	5633,8	5848,8	5963,2	5855,8	67,3	+/- 17,7
Pico de Carga con Errores de Enlace	4287,5	5610,1	6101	5731	301,3	+/- 26,5
Tiempo de Convergencia de Nodo Nuevo	22	24	31	23	2,1	+/- 0,7
Tiempo de Convergencia Total	148	151,4	156	149	3,4	+/- 1,6
Tiempo de Recuperación ante Caída de Enlaces	10	14,5	34	11	5,9	+/- 1,8
Tiempo de Recuperación ante Caída de Nodos	10	17	30	15	5,9	+/- 1,5

Tabla 7.10 OLSR Escenario aleatorio V

7.3.1 Conclusiones OSPF-MANET

Con un simple vistazo, se puede observar que los resultados obtenidos para los cinco escenarios son muy parecidos; esto indica que el protocolo OSPF-MANET se adapta muy bien a cualquier topología, lo que supone una gran ventaja en este tipo de redes, en las que ésta cambia constantemente y puede adoptar cualquier forma imaginable.
Estos resultados concuerdan con los vistos en la prueba anterior; puede verse que la carga de señalización por cada nodo en estado estable sigue siendo la misma, entre 11 y 12 Bytes/segundo, independientemente del número de nodos que exista en la topología, denotando un comportamiento muy eficiente. Ello es debido a que cada nodo intercambia información de señalización únicamente con sus nodos adyacentes.

En cuanto a la carga total en la red se aprecia, como es lógico, que ésta es, prácticamente, el doble con respecto a la prueba anterior, dado que hemos doblado el número de nodos. El valor obtenido es ligeramente superior al doble, lo que era de esperar, dado el problema de escalabilidad en estos protocolos, pese a que se intenta reducir mediante diferentes estrategias, como vimos en sus respectivos capítulos.
Lo mismo ocurre al observar los valores de carga máxima total en la red cuando se producen fallos en enlaces; en este caso, también se doblan los valores obtenidos respecto al experimento anterior (entre 270 y 320 Bytes/segundo en ésta última, frente a los 130 a 160 Bytes/segundo de la anterior).

En los tiempos de convergencia de nuevos nodos, los resultados son, una vez más, aproximadamente el doble, comparándolos con los de la prueba previa; es un resultado lógico, ya que tenemos el doble de nodos interviniendo en la configuración de la topología.
Estos tiempos, en este caso, están en torno a 37 segundos frente a los 18 de la prueba anterior, mientras que el tiempo de convergencia total hasta alcanzar el estado estable es de unos 200 segundos, resultado muy parecido, pero algo superior al caso anterior.
En esta situación, vemos que, a la hora de alcanzar el estado estable, el rendimiento del protocolo una vez escalada la red mejora, dado que el mecanismo de inundación, utilizado para intercambiar la información en este caso, está diseñado para mejorar también este aspecto, ya que se envían mensajes

broadcast, que son menos dependientes del número de nodos que formen parte de la red.

Por último, la medida de tiempos de recuperación frente a caídas de enlaces o nodos es similar, quizás ligeramente superior, influida por el aumento de nodos, pero prácticamente inapreciable, ya que en estos tiempos influye poco, por no decir nada, el número de nodos.

Mencionar que, de nuevo, se produce el ya citado problema de estabilidad.

Resumiendo, los resultados globales obtenidos siguen siendo excelentes, a pesar de haber pasado de nueve a veinte nodos, lo que indica un comportamiento muy bueno frente a la escalabilidad.
También es importante el comportamiento que se ha observado en los diferentes escenarios, ya que el tipo de topología existente no influye en los resultados. Esta característica es muy valiosa en este tipo de redes malladas inalámbricas, en las que la variedad de topologías es casi infinita, lo que aporta una robustez muy grande a la red, garantizando que no se generarán comportamientos extraños.

Por ahora, estos resultados convierten a OSPF-MANET en una opción muy a tener en cuenta, dada la poca carga introducida, su buena respuesta frente a la escalabilidad de la red y su uniformidad de comportamiento en diferentes topologías, todas ellas características claves en el tipo de redes bajo estudio.

7.3.2 Conclusiones OLSR

Los resultados muestran una carga media de señalización mucho mayor, comparándola con la obtenida en el experimento anterior con OSPF-MANET, concretamente, alrededor de 20 veces superior, tanto en las cargas medidas en nodos, como en las totales de la red.
Este resultado era de esperar, ya que OLSR genera una cantidad de información de control periódica mayor y más abundante; a causa de esto, la carga aumenta considerablemente.

Otro resultado importante es la reducción del retardo en todos los casos medidos, debido a que OLSR siempre dispone de rutas alternativas para suplir a otra que haya dejado de estar disponible por el motivo que sea; en el caso de OSPF-MANET, este proceso de cálculo comenzará al detectarse un cambio en la red, de ahí que los retardos introducidos para OSPF-MANET sean mayores.
Concretamente los retardos producidos por OLSR en estos escenarios son del orden del 25% inferiores a los obtenidos con OSPF-MANET.

Es necesario destacar el perfecto funcionamiento de la implementación de OLSR, que en ningún momento presentó problemas, ejecutándose correctamente durante todas las pruebas realizadas, frente al problema de estabilidad descubierto en el experimento con OSPF-MANET, que se mencionó en su momento.

Aunque se ha comprobado un aumento en la carga, éste puede ser asumido, dado que el ancho de banda real ofrecido por los estándares WiFi actuales [WIFI] es más que suficiente para admitir este volumen.
El posible problema surgiría, en el caso de utilizar un estándar WiFi 802.11g o inferior [WIFI], si el número de nodos se aumentara considerablemente, es decir, alrededor del triple actual (unos sesenta); en tal caso, podría comprometerse el rendimiento de la red, al saturarse algunos nodos con trafico de señalización, si la red no estuviera correctamente dimensionada y el tráfico correctamente distribuido, lo que sería fácilmente evitable contando con un sistema de control de sobrecarga

que detectara cuándo una ruta está ocupada en exceso y repartiera parte del trafico por rutas alternativas.

PARTE IV
CONCLUSIONES FINALES Y TRABAJOS FUTUROS

Capítulo 8 Conclusiones Finales y Trabajos Futuros

8.1 Introducción

Una vez finalizadas las pruebas, y teniendo ya los resultados para los dos experimentos, es posible desarrollar más en profundidad el análisis.
En concreto, se evaluarán ambos protocolos teniendo en cuenta, además de las propias medidas, otros aspectos no tan mensurables que conviene tener en cuenta para completar el estudio, como son:

- Soporte ofrecido por los desarrolladores

A la hora de decidir por una u otra opción es importante conocer si la escogida seguirá siendo funcional con el paso del tiempo; por ejemplo, dado que la implementación de MANET para OSPF está basada en el software de enrutamiento Quagga, es clave conocer si será adaptado para trabajar con las nuevas versiones que puedan ir apareciendo.

- Complejidad de instalación y configuración

Otro aspecto importante está en la dificultad de instalación y configuración de las distintas implementaciones. Es recomendable que la opción elegida sea muy sencilla en este aspecto, permitiendo crear un paquete de instalación que facilite al máximo la tarea, dado que el número de nodos, tanto en las pruebas como en un caso real, es elevado y que es necesario ofrecer a los diferentes usuarios finales una forma sencilla de instalación.

- Consumo de recursos

Como ya se ha mencionado en capítulos anteriores, dada la capacidad limitada de los dispositivos inalámbricos, es verdaderamente importante que la solución elegida administre estos recursos de una manera eficiente y reduciendo al máximo su consumo, con especial atención al uso de CPU, al de memoria y al gasto energético. Además, se ha de tener en cuenta el espacio de almacenamiento (memoria Flash y RAM disponible), ya que, por ejemplo, en los routers empleados en los experimentos, este espacio es muy limitado y es importante disponer de suficiente espacio libre, por si en un futuro fuera necesario añadir alguna nueva funcionalidad a los equipos.

Una vez terminadas las conclusiones finales, se realizará una breve descripción sobre las futuras líneas de trabajo que podrían seguirse utilizando este estudio como base.

8.2 Conclusiones Finales

Puesto que el objetivo principal de los experimentos ha sido obtener medidas representativas de la carga introducida y los retardos generados por estos protocolos, realizaremos a continuación un análisis profundo de los mismos, junto con otros aspectos a tener en cuenta, para el caso de los escenarios con topología aleatoria.

8.2.1 Carga

Los resultados de carga de señalización obtenidos en las topologías aleatorias, con y sin cambios o errores, son favorables para el caso de OSPF-MANET, con mediciones de 12 bytes/segundo por nodo y 200 bytes/segundo totales en la red, en el caso de no producirse cambios en la topología. Estos valores son excelentes comparándolos con los medidos para OLSR, 300 bytes/segundo por nodo y 6000 bytes/segundo totales en la red.

Por otra parte, en el caso de producirse cambios en la topología, OSPF-MANET experimenta un aumento de la carga, pasando a 300 bytes/segundo totales en la red, OLSR mantiene sus 6000 bytes/segundo.
Este comportamiento es debido al modo de funcionamiento de los protocolos, mientras que OSPF-MANET intercambia periódicamente una mínima cantidad de información de estado, realizando el proceso de cálculo de nuevas rutas sólo en el momento en que se genera un cambio en la red, OLSR intercambia mensajes de control con mayor frecuencia, una frecuencia que no se ve afectada por los cambios en la red, ya que simplemente eliminará las entradas obsoletas de sus tablas tras varios periodos sin que estas se actualicen.

Por estos motivos OSPF-MANET presenta una carga de señalización inferior a OLSR que, aunque aumenta cuando se producen cambios en la topología, se mantiene en valores inferiores a OLSR, que no presenta variaciones en este aspecto.
Además, ha de tenerse en cuenta que la utilización de interfaces inalámbricas conlleva un consumo energético elevado; por lo tanto, cuanto menos tráfico se envíe a través de estas interfaces, menor será el consumo realizado.
Por este motivo y los mencionados anteriormente, el protocolo OSPF-MANET se erige en claro vencedor en cuanto a consideraciones de carga se refiere.

Carga	**OSPF-MANET**	**OLSR**
Carga por nodo sin cambios en la red	12	300
Carga total sin cambios en la red	200	6000
Carga máxima con cambios en la red	300	6000

Tabla 8.1 Carga de señalización aproximada introducida a la red en bytes/segundo para 20 nodos

8.2.2 Retardo

El siguiente parámetro a tener en cuenta es el retardo introducido por los protocolos. En el caso de OSPF-MANET, este tiempo es superior comparándolo con los obtenidos para OLSR.

Para OSPF-MANET en los escenarios con topología aleatoria, tenemos que el retardo al introducir un nodo nuevo a la red es de aproximadamente 37 segundos, mientras que para OLSR es de unos 23 segundos. Esta tendencia se repite para el resto de retardos medidos, por lo que se concluye que OLSR introduce unos retardos bastante inferiores a OSPF-MANET.

Siguiendo con el retardo hasta alcanzar el régimen estacionario, se obtienen valores cercanos a 200 segundos para OSPF-MANET y unos 150 segundos para OLSR.

En cuanto al retardo a la hora de generarse una ruta alternativa cuando la principal deja de estar disponible a causa de la caída de un enlace entre dos nodos, OSPF-MANET obtiene tiempos de aproximadamente 20 segundos frente a los 15 de OLSR.

Por último, el retardo a la hora de generar nuevas rutas cuando es un nodo el que deja de estar disponible, es de 35 segundos mientras que para OLSR obtenemos valores cercanos a 18 segundos.

Estos resultados son debidos, de nuevo, al funcionamiento especifico de ambos protocolos. Mientras que OSPF-MANET comenzará el proceso de cálculo de rutas alternativas al producirse un cambio en la red, OLSR, dispondrá, casi instantáneamente, de una alternativa, por lo que el tiempo de retardo introducido será mucho menor.

Por ello el protocolo OLSR sale ganador en cuanto a consideraciones de retardo.

Retardo	**OSPF-MANET**	**OLSR**
Retardo al introducir un nodo nuevo a la red	37	23
Retardo en alcanzar el régimen estacionario	200	150
Retardo de recuperación en cambios en enlaces	20	15
Retardo de recuperación en cambios de nodos	35	18

Tabla 8.2 Retardos introducidos por los protocolos, en segundos, para el caso de 20 nodos

8.2.3 Soporte de los Desarrolladores

Considerando el soporte por parte de los desarrolladores, la implementación utilizada en este proyecto para OSPF-MANET lleva sin actualizarse desde el año 2008 como puede verse en su página web [PARCHEMDR], y la última versión de Quagga [QUAGGA] para la que se aplica el parche es la 0.99.9, mientras que dicho software, a fecha de realización de este estudio, se encuentra ya por la 0.99.16.

Esto supone una desventaja clara respecto a la implementación de OLSR [OLSRD] ya que este último se encuentra en constante evolución y cuenta con una comunidad de usuarios y desarrolladores muy extensa y activa.
Por este motivo, mientras que OSPF-MANET comienza a quedar obsoleto, OLSR sigue mejorando su funcionalidad y actualizándose muy frecuentemente, resultando claramente la mejor opción con vistas al futuro.

8.2.4 Configuración e Instalación

Desde el punto de vista de configuración e instalación, OLSR es de nuevo la mejor opción, ya que, simplemente con instalar el paquete disponible queda totalmente funcional y listo para ejecutarse. En cambio, para OSPF-MANET el proceso es bastante más complejo, teniendo primero que aplicar el parche al software de Quagga, compilarlo, e instalar varios demonios y librerías, además de tener que modificar varios parámetros en su fichero de configuración.

8.2.5 Consumo de Recursos

En cuanto al consumo de recursos de ambas implementaciones, OSPF-MANET sale ganando. El funcionamiento del protocolo OLSR provoca un uso elevado de la interfaz inalámbrica, lo que conlleva un consumo energético superior y un coste computacional también superior, a causa del constante intercambio y procesado de paquetes de señalización.

Por el contrario, el protocolo OSPF-MANET, gracias a su particular sistema, reduce tanto el uso de la interfaz inalámbrica como el intercambio de mensajes de

encaminamiento, gestionando los recursos de un modo más eficiente. Pese a que OLSR cuenta con el sistema de inundación a través de MPRs, visto en la sección 4.3, para reducir la cantidad de información de señalización generada, el mecanismo empleado por OSPF-MANET sigue resultando más eficiente en este aspecto; esto ha sido comprobado mediante la medición del consumo de CPU y memoria con el comando *top*, durante la ejecución de las diferentes pruebas.

8.2.6 Estabilidad

Por último, un aspecto muy importante a tener en cuenta es la estabilidad de las implementaciones utilizadas. En el caso de OSPF-MANET, se han detectado problemas que provocan la caída aleatoria de los demonios, obligando a rearrancarlos manualmente; por el contrario, la ejecución de OLSR ha sido perfecta en todas las pruebas realizadas, no produciéndose fallos en ningún momento.
Este problema podría solucionarse utilizando otra implementación de OSPF-MANET, pero en el caso que nos ocupa ha sido imposible solucionarlo.

8.2.7 Otras Características

Otras ventajas del protocolo OLSR frente a OSPF-MANET son:

- La facilidad de añadir funcionalidad extra mediante el uso de plugins sin alterar el funcionamiento básico del protocolo (visto en la sección 4.4.3), ya que se diseñó pensando en este fin. Dado que la utilización de las redes malladas inalámbricas es reciente, esta capacidad permitiría su sencilla adaptación a nuevas características que pudieran requerirse.

- Está ideado para trabajar independientemente con otros protocolos (sección 4.1), por lo que no influirá en su funcionamiento, pudiéndose ejecutar en paralelo.

- Cualquier aplicación podría utilizar el método de inundación de OLSR para enviar mensajes a través de la red (sección 4.4.3). De este modo el demonio olsrd se ejecutaría como agente de retransmisión para aplicaciones locales y así reducir la carga de la red.

8.2.8 Resumen

En resumen, las principales ventajas de OSPF-MANET son la mínima carga introducida en la red y un consumo menor de recursos, aspectos fundamentales en el tipo de redes analizadas; en el resto de aspectos, en cambio, es muy inferior a OLSR.
A pesar de que la carga en OLSR es elevada, la tecnología inalámbrica actual está de sobra capacitada para soportar estas tasas de transferencia.

A pesar de hacer un gasto superior de recursos comparándolo con OSPF-MANET, globalmente no es demasiado elevado, pudiendo ser soportado perfectamente por los dispositivos limitados existentes en la actualidad

Por otro lado, la gran estabilidad de la implementación de OLSR proporciona un mecanismo eficaz y fiable para gestionar el encaminamiento en las redes malladas inalámbricas.

La facilidad de instalación y configuración de OLSR proporciona una gestión sencilla de los nodos de la red y ofrece a los usuarios finales una manera muy simple para su utilización.

En otro ámbito, el soporte ofrecido por sus desarrolladores, la gran comunidad de usuarios de que dispone, los habituales eventos celebrados por ésta para fomentar la interacción, las continuas mejoras y actualizaciones garantizan que OLSR es una opción de presente y de futuro.

Característica	**OSPF-MANET**	**OLSR**
Carga		
Retardo		
Soporte		
Instalación y configuración		
Consumo de recursos		
Estabilidad		

Tabla 8.3 Comparativa de características entre OSPF-MANET y OLSR

Concluyendo esta comparativa se puede decir, sin lugar a dudas, que la implementación de OLSR es la mejor opción como protocolo de encaminamiento para las redes malladas inalámbricas, ya que, a pesar de que OSPF-MANET obtiene mejores resultados en aspectos muy importantes como la carga y el consumo de recursos, la inestabilidad de la implementación estudiada y el resto de ventajas que posee OLSR frente a su competidora, convierten a OLSR en la opción más apropiada.

8.3 Futuras Líneas de Trabajo

Como se ha visto a lo largo de este estudio, el objeto de esta memoria se ha centrado en el análisis de los protocolos OSPF-MANET y OLSR en su utilización para redes malladas inalámbricas.
Sin embargo, el ámbito de este tipo de redes es tan amplio que permite la extensión de este estudio, con el fin de ser utilizado como base de futuros experimentos, algunos de los cuales serán citados a continuación.

8.3.1 Extensión a FloorNet

Una de las posibilidades más inmediatas consiste en extender los escenarios de pruebas a una red FloorNet, como la que ya se encuentra desplegada en uno de los laboratorios de la Universidad [FLOORNET].
Este tipo de redes está formado por una red inalámbrica multisalto cuyos nodos se encuentran ubicados bajo el falso suelo, aprovechando de este modo el espacio disponible para ocultarlos y protegerlos. Además, cuentan con una infraestructura que proporciona tanto la alimentación necesaria como conectividad cableada si fuera precisa.

La figura 8.1[5] muestra el falso suelo del laboratorio, en el que se pueden observar varios de los elementos.

[5] Fuente de la imagen: http://floornet.org/false_floor_small.jpg

Figura 8.1 Ejemplo de FloorNet

8.3.2 Extensión del Estudio a Otros Protocolos

Aunque para este estudio se han elegido OSPF-MANET y OLSR, existen una amplia variedad de protocolos de encaminamiento que pueden ser utilizados sobre las redes malladas inalámbricas, como son BATMAN [BATMAN], AODV [AODV], DSR [DSR], etc. con los que se podría poner en práctica la misma metodología aquí empleada.

8.3.3 Ampliación del Número de Interfaces en Cada Nodo

Este estudio se ha centrado en la utilización de nodos con una interfaz inalámbrica única; otra posibilidad sería utilizar otro modelo de router que disponga de más de una interfaz inalámbrica, con el fin de profundizar en el comportamiento de los protocolos de encaminamiento a la hora de enrutar tráfico a través de nodos con múltiples interfaces inalámbricas.

Parte V
APENDICES

APENDICE A – Instalación de la Implementación de OSPF-MANET y de OLSR en PC

Preparación de la implementación del protocolo OSPF-MANET para su instalación, en PC.
Para ello, hay que descargar todo lo necesario de su página web[6], donde se pueden encontrar tanto el parche necesario para añadir la funcionalidad requerida para las MANETs, como la versión de Quagga para la que esta diseñado dicho parche.

- Parche OSPFv3 MANET MDR 1.01 patch (2 de Junio de 2008)

Encargado de añadir la funcionalidad necesaria para el correcto funcionamiento del protocolo OSPF en MANETs al software de enrutamiento Quagga.

- Quagga 0.99.97

Software de enrutamiento que permite convertir cualquier máquina, que disponga de un sistema operativo basado en Linux, en un router.

El siguiente paso consistió en aplicar el parche a la versión correcta de Quagga.

A.1 Parchear el Software de Quagga

Descomprimir Quagga:

tar xvfz quagga-0.99.9.tar.gz

Situarse en la carpeta creada:

cd quagga-0.99.9/

Aplicar el parche:

patch -p1 < ../quagga-0.99.9.ospfv3-manetmdr.patch

Previamente a la instalación en sí, es necesario instalar una serie de librerías y aplicaciones que son requisito imprescindible, sin las cuales el proceso de instalación no podría ser completado. Son las siguientes: *autoconf 2.13, gawk, libtool, libreadline5-dev* y *dpkg-dev*.

A.2 Requisitos Previos para la Instalación de Quagga

sudo apt-get install autoconf2.13

sudo apt-get install gawk

sudo apt-get install libtool

sudo apt-get install libreadline5-dev

sudo apt-get install dpkg-dev

[6] http://hipserver.mct.phantomworks.org/ietf/ospf/
[7] http://www.quagga.net/

Una vez preparado todo, se procede a la instalación que requiere la ejecución previa de unos comandos que prepararán y habilitaran diferentes opciones.

A.3 Ejecución de Comandos Previos a la Instalación

```
./update-autotools

./configure --enable-user=root --enable-group=root --enable-vtysh --with-
cflags=-ggdb --prefix=/tmp/usr

make
```

La ejecución del make pude dar un fallo, para solucionarlo es necesario introducir el siguiente comando tras el error y volver a ejecutar make.

```
awk -f ./memtypes.awk ./memtypes.c > memtypes.h
```

Una vez ejecutados ya puede realizarse la instalación.

A.4 Instalación de Quagga

```
sudo make install
```

En ocasiones al modificar al instalar un fichero de configuración donde están las librerías (en este caso /etc/ld.so.conf.d/libc.conf) este proceso no se actualiza automáticamente y se ha de realizar a mano ejecutando:

```
ldconfig
```

Tras la instalación, es necesario copiar los ficheros de configuración de Quagga a su carpeta correspondiente y renombrarlos con la denominación apropiada para su correcto funcionamiento.

A.5 Preparación de los Ficheros de Configuración

```
sudo cp zebra.conf.sample zebra.conf

sudo cp ospf6d.conf.sample ospf6d.conf

sudo cp ospfd.conf.sample ospfd.conf
```

Cuando todo esté preparado, se arrancarán los demonios necesarios, en nuestro caso *zebra* y *ospf6d*.

A.6 Arrancando los Procesos Necesarios

```
sudo /usr/local/sbin/zebra -d

sudo /usr/local/sbin/opfd -d
```

Finalmente, con los procesos ejecutándose, se lanza *vtysh*, una shell que integra todas las funcionalidades del software de routing Quagga y permite un control total sobre los distintos demonios.

A.7 Ejecución de la Shell

```
sudo vtysh
```

Para probar el correcto funcionamiento básico del protocolo de enrutamiento OSPF, configuramos todo lo necesario en los dos equipos mediante la shell *vtysh*.

A.8 Configurar OSPF entre 2 Máquinas Usando Quagga

carmen01 es el PC número 1 y carmen02 el número 2.

Para comprobar que OSPF funciona correctamente, necesitamos emplear rutas que no estén directamente conectadas, para ver si se envían los anuncios a través de los demás enlaces; para ello, debemos desconectar el cable de red para las interfaces eth1 de ambos equipos, que los interconectan entre sí, y asignar dichas interfaces a dos redes diferentes, para observar si éstas se anuncian a través de los eth0, que en ambos equipos se encuentran conectadas a la red del laboratorio, como debería suceder si funcionara correctamente.

```
carmen01# conf term
carmen01(config)# inter eth1
carmen01(config-if)# ip addr 10.0.0.1/24
carmen01(config-if)# end
carmen01# conf term
carmen01(config)# router ospf
carmen01(config-router)# network 10.0.0.0/24 area 0
carmen01(config-router)# network 163.117.140.0/24 area 0
carmen01(config-router)# end
carmen01# show ip route
carmen01# show ip ospf
carmen01# show ip ospf database
carmen01# show ip ospf neigh

carmen02# conf term
carmen02(config)# inter eth1
carmen02(config-if)# ip add 20.0.0.1/24
carmen02(config-if)# end
carmen02# conf term
carmen02(config)# router ospf
carmen02(config-router)# network 20.0.0.0/24 area 0
carmen02(config-router)# network 163.117.140.0/24 area 0
carmen02(config-router)# end
carmen02# show ip route
carmen02# show ip ospf
carmen02# show ip ospf database
carmen02# show ip ospf neigh
```

El siguiente paso consiste en probar la funcionalidad básica del protocolo OLSR; al no necesitar ningún requisito previo ni ninguna modificación extra, basta con instalarlo mediante el gestor de aplicaciones synaptic.

Para probar su funcionamiento, debemos configurar las interfaces eth1 de ambos equipos con una dirección global IPv6 perteneciente a redes diferentes (apéndice A.9) y arrancar el protocolo mediante su comando correspondiente en ambos equipos (apéndice A.10).

A.9 Comprobando el Funcionamiento de OLSR

Al igual que ocurría con OSPF (apéndice A.8), para comprobar que OLSR funciona correctamente necesitamos emplear rutas que no estén directamente conectadas, para ver si se envían los anuncios a través de los demás enlaces, por lo que debemos desconectar el cable de red para las interfaces eth1 de ambos equipos, que los interconectan entre sí, y asignar dichas interfaces a dos redes diferentes, para observar si éstas se anuncian a través de los eth0, que en ambos equipos se encuentran conectadas a la red del laboratorio, como debería suceder si funcionara correctamente.

En carmen01

sudo ip addr add 3ffe:1::1/128 dev eth1

En carmen02

sudo ip addr add 3ffe:2::1/128 dev eth1

A.10 Arrancando el Demonio olsrd

olsrd

Resumiendo, tras solventar los problemas surgidos a la hora de instalar Quagga, podemos concluir que ambos protocolos funcionan correctamente al probar su funcionalidad básica. OLSR es más sencillo de instalar y configurar, ya que no necesita de la instalación de ningún módulo adicional por estar diseñado específicamente para las redes MANET, mientras que OSPF-MANET es más complejo de instalar y algo más engorroso a la hora de configurar, comparándolo con la sencillez de OLSR.

APENDICE B - Compilación del Software de Routing Quagga para el Entorno Openwrt

Para conseguir una versión funcional de la aplicación Quagga con el parche de MANET, se intentaron diferentes posibilidades, varias de ellas fallidas, hasta encontrar la solución correcta.

Hay que mencionar que existe una versión de Quagga desarrollada específicamente para el entorno Openwrt[8], y en concreto para la versión instalada en los routers (kamikaze 8.09 para el chip broadcom 47xx), pero no incluye el parche de MANET necesario para desarrollar las pruebas, por lo que no es posible utilizar esta versión directamente y ha de compilarse una propia que lo incluya, valiéndose del SDK correspondiente.

El primer intento consistió en seguir directamente el tutorial oficial para compilar software disponible en la página oficial de Openwrt[9], usando la versión de Quagga más cercana a la diseñada para el parche[10], ya que la propia 0.99.9 no estaba disponible en su versión para Openwrt debido a su antigüedad; en concreto, la versión probada fue la 0.98 stable.
Para su correcta compilación, fue necesaria la instalación de una serie de librerías y aplicaciones (apéndice B.1), así como la modificación del fichero *makefile*[11] (apéndice B.2).

B.1 Instalando las Librerías Necesarias

```
sudo apt-get install libncurses5-dev ncurses-term zlib1g-dev gawk bison flex
autoconf subversion g++
```

B.2 Modificaciones en el Fichero makefile

```
PKG_VERSION:=0.99.9

Comando para obtener el nuevo MD5

md5sum quagga-0.99.9.tar.gz

PKG_MD5SUM:=(MD5 obtenido con el comando anterior)
```

A continuación, se procedió a la instalación del SDK[12].

B.3 Instalación del SDK

```
cd /tmp
wget http://downloads.openwrt.org/whiterussian/newest/OpenWrt-SDK-
Linux-i686-1.tar.bz2
bzcat OpenWrt-SDK-Linux-i686-1.tar.bz2 | tar -xf -
cd OpenWrt-SDK-Linux-i686-1/package/
```

[8] http://downloads.openwrt.org/kamikaze/8.09/
[9] http://oldwiki.openwrt.org/BuildingPackagesHowTo.html
[10] https://dev.openwrt.org/browser/trunk/openwrt/package/quagga?rev=980
[11] Fichero que contiene las instrucciones de compilación relacionadas con la utilidad make, facilitando la compilación de programas.
[12] http://downloads.openwrt.org/whiterussian/newest/OpenWrt-SDK-Linux-i686-1.tar.bz2

```
svn co https://svn.openwrt.org/openwrt/trunk/openwrt/package/atftp/
cd ..
make atftp-clean && make atftp-compile

ls -al bin/packages
-rw-r--r-- 1 openwrt-dev openwrt-dev 18249 2006-01-24 23:03 atftp_0.7-
1_mipsel.ipk
-rw-r--r-- 1 openwrt-dev openwrt-dev 31819 2006-01-24 23:03 atftpd_0.7-
1_mipsel.ipk
```

Se crearon los directorios requeridos por el tutorial y se copiaron los ficheros a sus carpetas correspondientes y, a continuación, se comenzó la compilación.

B.4 Crear los Directorios

```
cd ~/OpenWrt-SDK-Linux-i686-1
mkdir -p package/quagga/ipkg
mkdir -p package/quagga/patches

Mover el archivo de configuración
package/quagga/Config.in
Mover el fichero makefile a su carpeta correspondiente
package/quagga/Makefile
Mover también los archivos control:

package/quagga/ipkg/Ficheros.control

Comando para iniciar la compilación

make clean && make V=99
```

Este proceso produjo un error, debido a un problema con la estructura de carpetas, ya que creaba dos subdirectorios con el mismo nombre, uno dentro del otro, cuya solución fue eliminar uno de estos subdirectorios repetidos.

B.5 Mensaje de Error

```
/bin/bash ../libtool --tag=CC   --mode=link mipsel-linux-uclibc-gcc  -ggdb  -
o testzebra test_main.o zebra_rib.o interface.o connected.o debug.o
zebra_vty.o kernel_null.o redistribute_null.o ioctl_null.o misc_null.o
../lib/libzebra.la -lcrypt
libtool: link: mipsel-linux-uclibc-gcc -ggdb -o .libs/testzebra test_main.o
zebra_rib.o interface.o connected.o debug.o zebra_vty.o kernel_null.o
redistribute_null.o ioctl_null.o misc_null.o  ../lib/.libs/libzebra.so -lcrypt
../lib/.libs/libzebra.so: undefined reference to `rpl_realloc'
../lib/.libs/libzebra.so: undefined reference to `rpl_malloc'
collect2: ld returned 1 exit status
```

Ya sabiendo el modo de funcionamiento del SDK, es posible aplicar el parche de MANET a la versión organizada en las carpetas del SDK, para posteriormente compilarla.
Dados los problemas generados por este primer intento, se consideró la posibilidad de utilizar un emulador de router llamado QEMU, que permite emular imágenes del

firmware de Openwrt siguiendo el tutorial oficial de Openwrt[13] utilizando una de las imágenes listas para instalar de la página de Openwrt[14].
Finalmente, se desestimó esta posibilidad por no ser viable a la hora de realizar las pruebas en los escenarios.

La solución definitiva consistió en realizar un backport (hacer un cambio de versión de una más moderna a otra más antigua). Se descargó la última versión estable de Quagga disponible en el repositorio de aplicaciones de Openwrt, en concreto la 0.99.12, y siguiendo los pasos descritos en el primer intento, o lo que es lo mismo, organizar los archivos de Quagga en las carpetas creadas en el SDK, aplicar el parche de MANET sobre estos ficheros, modificar el fichero *makefile* correspondiente y finalmente compilar con el comando destinado para ese fin.

[13] http://oldwiki.openwrt.org/RunningKamikazeOnQEMUHowTo.html
[14] http://downloads.openwrt.org/kamikaze/8.09/x86/

APENDICE C - Compilación de Firmware Openwrt con Funcionalidad Mínima

A causa del problema mencionado en el apéndice anterior, es imprescindible compilar una versión propia del firmware Kamikaze 8.09 de Openwrt con la funcionalidad mínima que garantice la completa realización de las pruebas.

Para esta tarea, lo primero será descargar el código fuente del firmware sin compilar[15].
Tras descomprimirlo, ejecutaremos su utilidad de configuración, desde la cual se podrá modificar diferentes parámetros, así como seleccionar qué módulos o aplicaciones serán instalados.

C.1 Descomprimir el Firmware

tar xvfz kamikaze_8.09_source.tar.bz2

Arrancar la utilidad de configuración

make menuconfig

Hecho esto, nos queda establecer qué aplicaciones serán imprescindibles para la realización de las pruebas (apéndice C.2):

- Necesitaremos una utilidad que nos permita configurar las interfaces de red, por lo que instalaremos *ip*.
- Necesitaremos una utilidad que nos permita crear topologías sin tener que desplegarlas, por lo que instalaremos *iptables*.
- También nos interesa, dado que trabajaremos con IPv6, su correspondiente versión *ip6tables*.
- Como acabamos de mencionar, las pruebas se van a realizar sobre IPv6, por lo que tenemos que asegurarnos que esté soportado por el firmware; debido a esto, instalaremos el módulo *kmod-ipv6*, que añade soporte para IPv6.
- Por último, no podemos olvidar añadir un módulo que se encargue de habilitar la interfaz inalámbrica para el chip utilizado por el router; por lo tanto, instalaremos el driver *kmod-b43*.

Una vez configurado, se ejecutó el comando para compilar (apéndice C.3), dando como resultado varios firmwares diferentes para diferentes chips, en nuestro caso escogimos la versión generada para br47xx.

C.2 Captura de los Menús de Configuración del SDK

[15] http://downloads.openwrt.org/kamikaze/8.09/kamikaze_8.09_source.tar.bz2

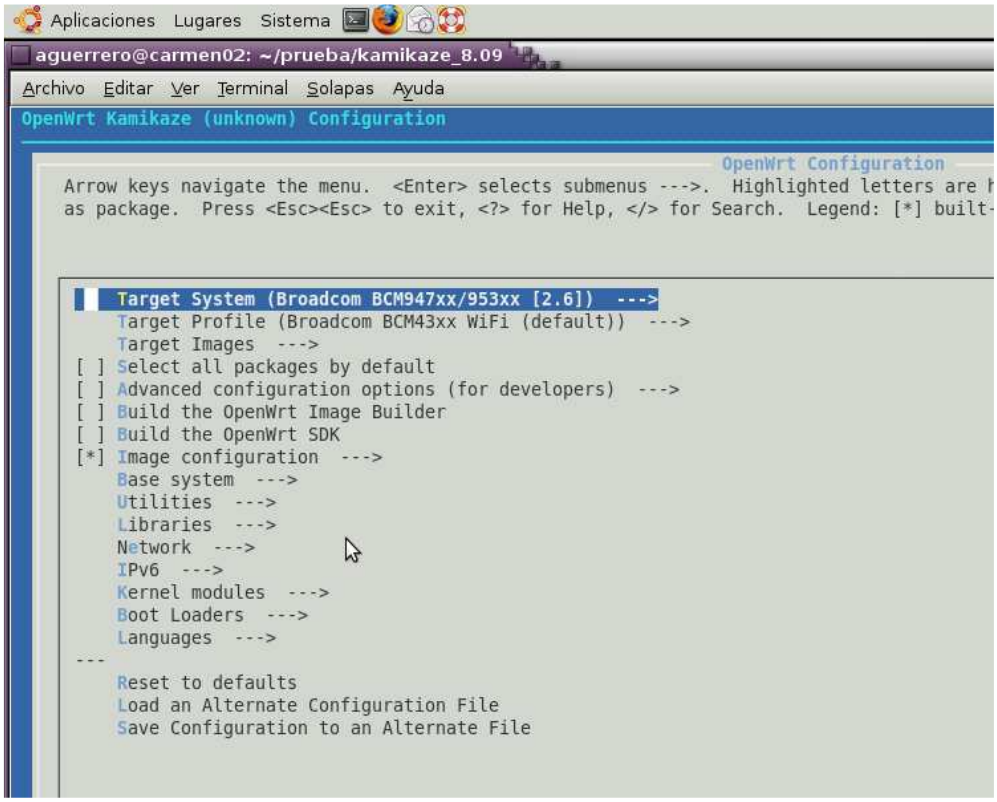


Figura 8.2 Vista del menú principal de la herramienta de creación de Openwrt

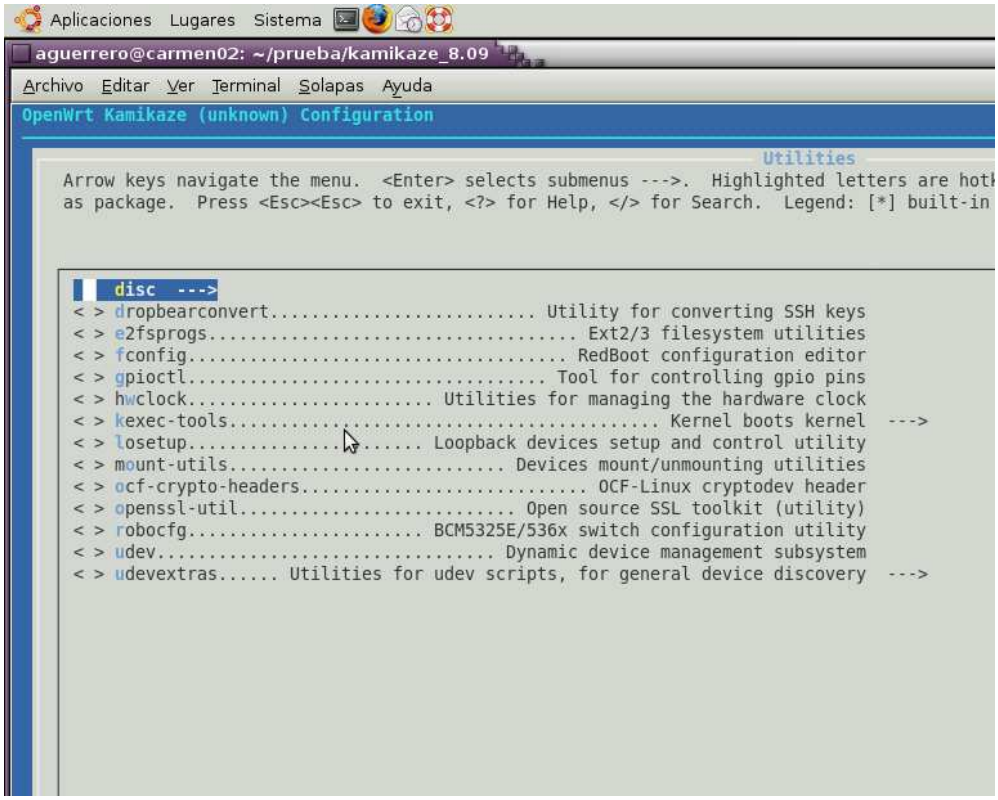


Figura 8.3 Vista del menú de utilidades

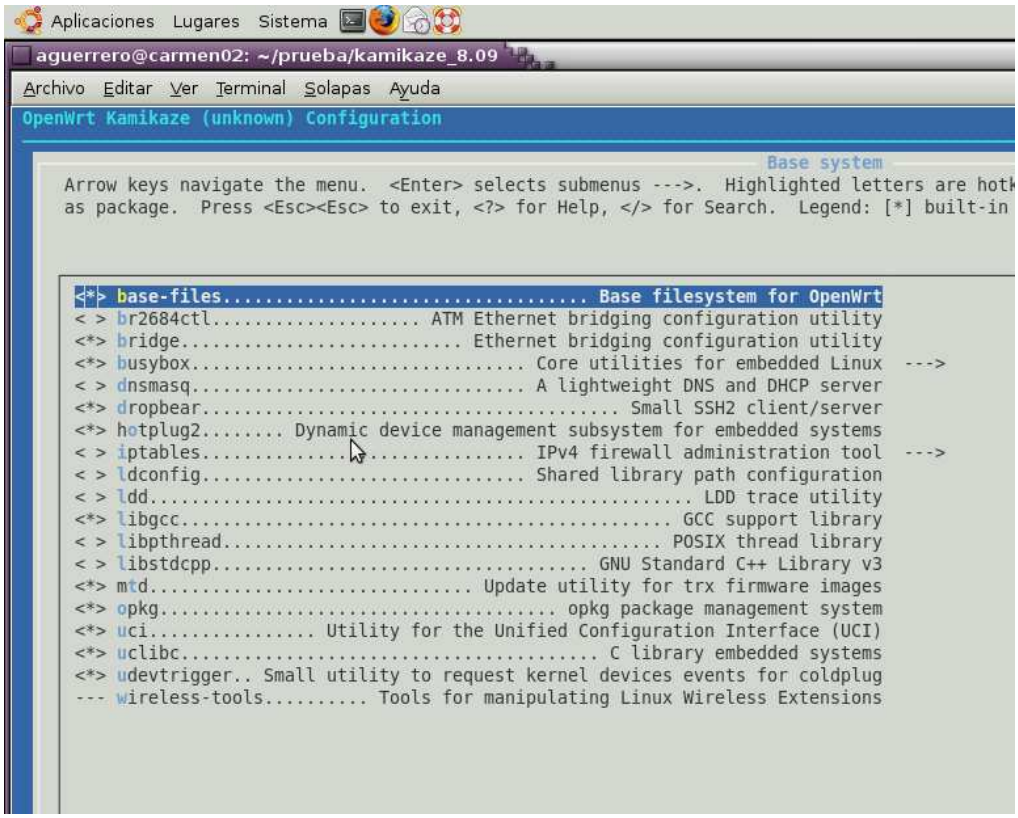


Figura 8.4 Vista del menú donde se seleccionan la base del firmware

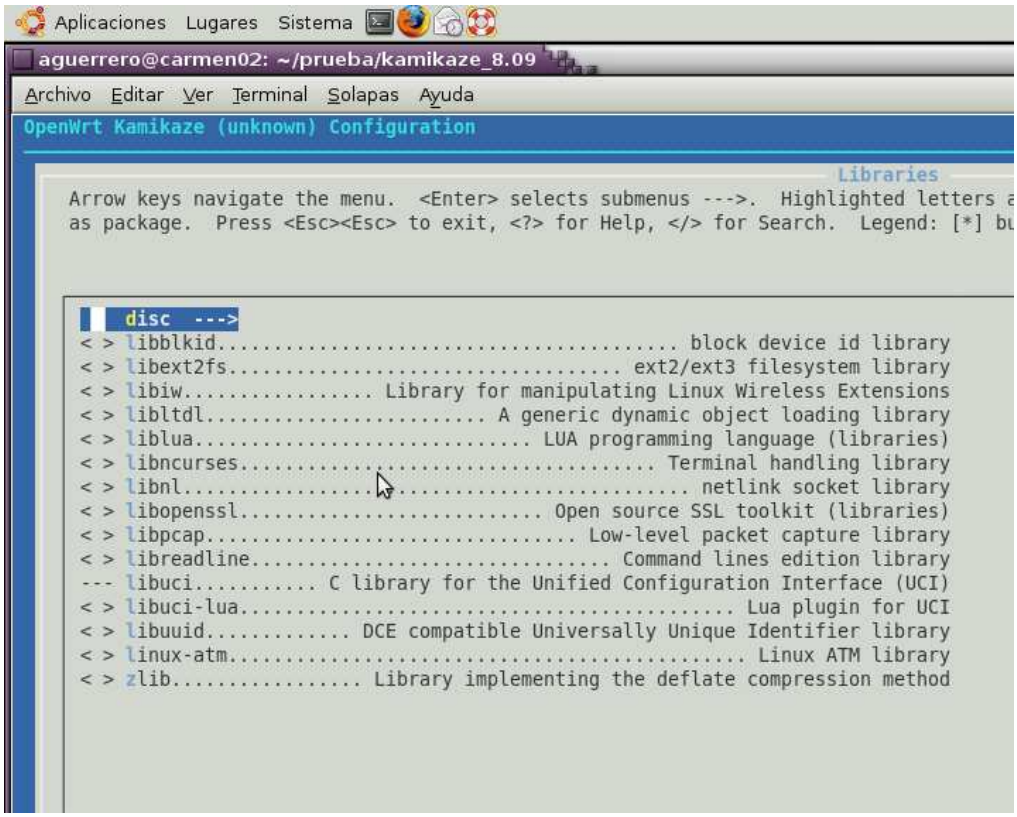


Figura 8.5 Vista del menú de librerías

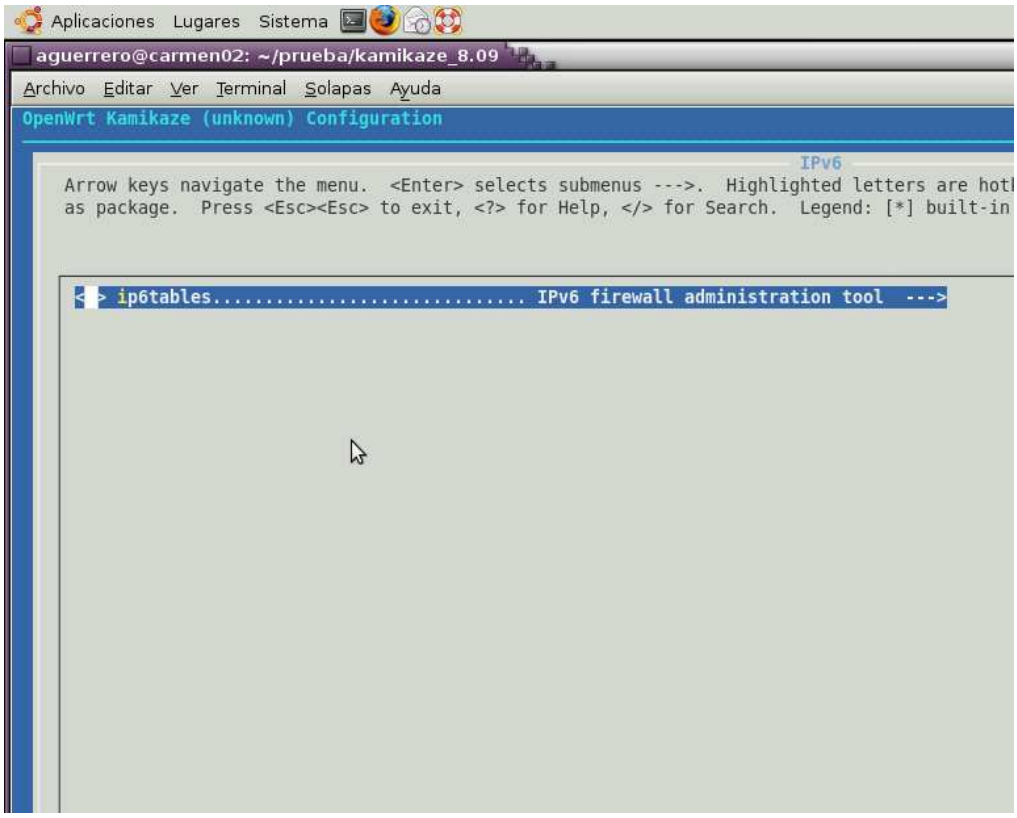

Aplicaciones Lugares Sistema
aguerrero@carmen02: ~/prueba/kamikaze_8.09
Archivo Editar Ver Terminal Solapas Ayuda
OpenWrt Kamikaze (unknown) Configuration
IPv6
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hot
as package. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in
< > ip6tables..... IPv6 firewall administration tool --->
Aplicaciones Lugares Sistema
aguerrero@carmen02: ~/prueba/kamikaze_8.09
Archivo Editar Ver Terminal Solapas Ayuda
OpenWrt Kamikaze (unknown) Configuration
Network
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters
as package. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*]
< > arptables..... ARP firewalling software
< > ebtables..... Ethernet bridge firewall administration utility
< > ebtables-utils..... ebtables save/restore utilities
< > genl..... General netlink utility frontend
< > hostapd..... IEEE 802.1x Authenticator (full)
< > hostapd-mini..... IEEE 802.1x Authenticator (WPA-PSK only)
< > ifenslave..... A utility to manipulate kernel ethernet bonding devices
<*> ip..... Routing control utility
< > ipset..... Netfilter ip sets administration utility
< > iw..... cfg80211 interface configuration utility
< > ppp..... PPP daemon
< > pptp..... PPTP client
< > tc..... Traffic control utility
< > wpa-supplicant..... WPA Supplicant

Figura 8.6 Vista del menú de utilidades para IPv6

Figura 8.7 Vista del menú de utilidades para redes

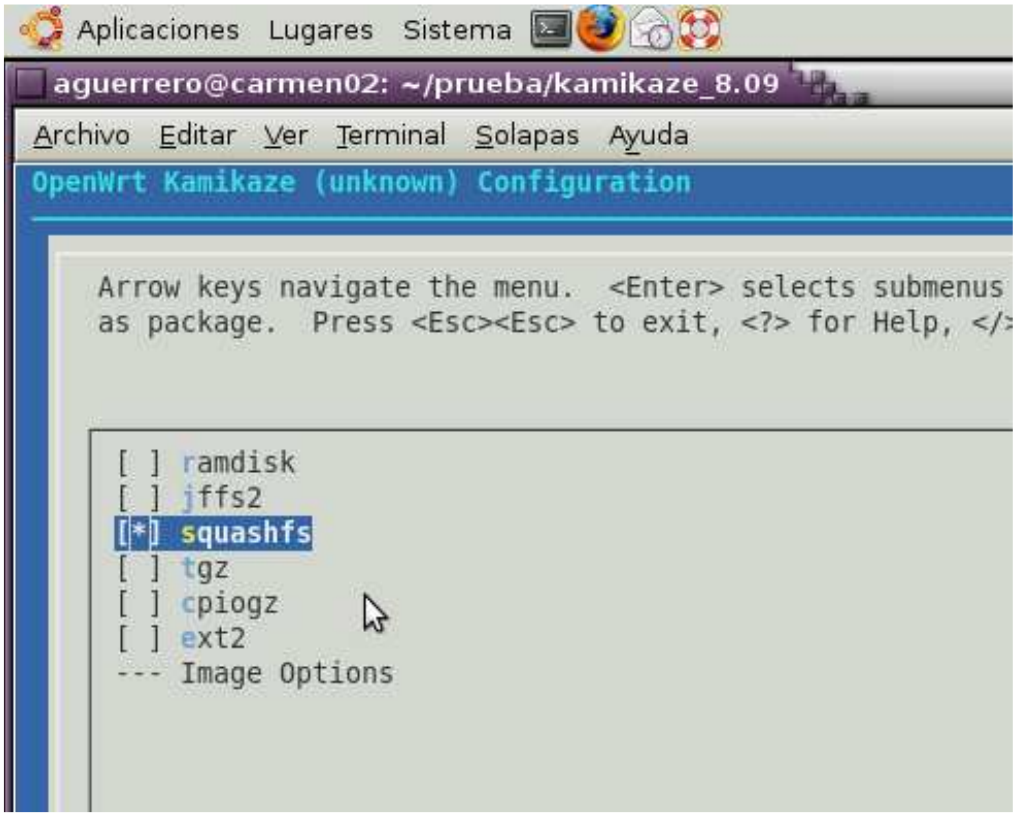


Figura 8.8 Vista del menú de selección de formato de imagen

C.3 Compilando el Firmware

Comando para compilar el firmware

make world

APENDICE D – Instalación y Configuración del Firmware Openwrt Kamikaze

Dado que los routers que se emplearon en las pruebas han sido usados en distintos proyectos, prácticas, etc., podrán encontrarse instalados diferentes tipos de firmware; por este motivo, cada router tendrá que ser previamente analizado, de una manera individual, para seleccionar el sistema óptimo con el que instalar el nuevo firmware.

Por suerte, todos los nodos disponían de firmwares Openwrt en distintas versiones, lo que facilitó mucho el trabajo. Hubo que usar dos métodos diferentes de instalación: el primero para los casos en los que se podía acceder directamente al router, porque se conocía la contraseña de acceso por ssh (apéndice D.1) y el segundo cuando no se conocía la contraseña y, por lo tanto, era imposible acceder al mismo (apéndice D.2).

Si se diera el caso de disponer de routers sin un firmware Openwrt se tendrá que seguir otro proceso distinto (apéndice D.3).

D.1 Flasheo del Firmware cuando el Router es Accesible por SSH

Copiar la imagen del firmware en el router

scp openwrt-brcm47xx-squashfs.trx root@IPdelRouter:/tmp/

Flashear el router y reiniciar

mtd write /tmp/openwrt-brcm47xx-squashfs.trx linux && reboot

Para poder acceder al router una vez flasheado

telnet 192.168.1.1

D.2 Flasheo del Firmware cuando el Router no es Accesible por SSH[16] (Modo FailSafe)

Desconectar la alimentación del router

Conectar el cable de red en la toma LAN1

Asignar una dirección al PC en la subred 192.168.1.0/24

Conectar el cable de alimentación y esperar a que el LED DMZ luzca

En cuanto luzca presionar cualquier botón (Reset o Secure Easy Setup) varias veces

Si se realiza correctamente el LED DMZ lucirá intermitentemente 3 veces por segundo

En ese momento será posible acceder al router mediante un telnet a la dirección 192.168.1.1

16

http://oldwiki.openwrt.org/OpenWrtDocs%282f%29Hardware%282f%29Linksys%282f%29WRT54GL.html

Para flashear el nuevo firmware ejecutar: mount_root para permitir la escritura

Ejecutar el comando passwd para habilitar el uso de SSH

Una vez hecho esto se podrá utilizar el método descrito en D.1

D.3 Flasheo del Firmware en Routers sin Firmwares Openwrt Previo

Importante recordar que previo a la instalación del firmware 2.6 hay que tener instalado el 2.4

Nos bajamos la imagen del firmware a instalar[17]

```
apt-get install tftp
tftp 192.168.1.1
tftp> binary
tftp> trace
tftp> put openwrt-wrt54g-squashfs.bin
```

Una vez instalado correctamente el firmware, es necesario seguir una serie de pasos para modificar algunos parámetros de configuración que, a causa de la nueva instalación, se quedan por defecto con un valor inválido.

El primer cambio consiste en habilitar el acceso por SSH (apéndice B.4), ya que, por defecto, sólo podrá accederse al router mediante telnet.

D.4 Habilitar el Acceso por SSH al Router

passwd

Introducir un usuario y una contraseña

Es posible que, al intentar acceder mediante SSH al router desde un PC, la clave RSA ya haya sido asignada para la dirección IP del router, por lo que es necesario eliminar del registro de claves la asignada a esa IP.

D.5 Eliminar del Registro de Claves RSA una Clave Asignada a una IP

ssh-keygen -R hostname|ip

El siguiente cambio será habilitar la interfaz inalámbrica que, por defecto, se encuentra deshabilitada.

D.6 Habilitar la Interfaz Inalámbrica en un Router

uci set wireless.wlan0.disabled=0

uci commit wireless && wifi

Dado que las pruebas utilizarán en todo momento el modo de conexión Ad-hoc en las interfaces inalámbricas, interesa que esto se establezca automáticamente cada vez que arranque el router, evitando de este modo tenerlo que configurar cada vez que se encienda.

[17] http://kamikaze.openwrt.org/8.09/brcm-2.4/openwrt-wrt54g-squashfs.bin

D.7 Habilitar la Configuración Automática del Modo Ad-hoc al Arrancar el Router

Para conseguir que se habilite el modo Ad-hoc al inicio, ha de copiarse el fichero S70adhoc en /etc/rc.d/

scp S70adhoc root@192.168.200.6:/etc/rc.d/

Cuyo contenido ha de ser el siguiente:

```
#!/bin/sh /etc/rc.common

ifdown wifi
iwconfig wlan0 mode Ad-hoc
ifup wifi
```

Uno de los cambios más importantes es la edición del fichero */etc/config/network* (apéndice D.8), que establecerá la configuración de las distintas interfaces del router; de este modo, nos interesa que las interfaces queden como sigue (donde X será el identificador asignado al router, por ejemplo, Router 1 -> 192.168.200.1, 192.168.2.1):

Puerto	**Dirección**
1:	192.168.200.X
2:	192.168.2.X
3	192.168.2.X
4	192.168.2.X
Int	DHCP (acceso a Internet)

Tabla 8.4 Direccionamiento en los diferentes puertos del router

D.8 Modificación del fichero Network

X será el identificador asignado al router, empezando a contar en 1

```
#### VLAN configuration
config 'switch' 'eth0'
    option 'vlan0' '0 1 2 5*'
    option 'vlan1' '3 5'
    option 'vlan2' '4 5'

#### Loopback configuration
config 'interface' 'loopback'
    option 'ifname' 'lo'
    option 'proto' 'static'
    option 'ipaddr' '127.0.0.1'
    option 'netmask' '255.0.0.0'

#### LAN configuration
config 'interface' 'lan'
    option 'ifname' 'eth0.0'
    option 'proto' 'static'
    option 'ipaddr' '192.168.2.X'
    option 'netmask' '255.255.255.0'

#### CONF configuration
config 'interface' 'conf'
```

```
    option 'ifname' 'eth0.1'
    option 'proto' 'static'
    option 'ipaddr' '192.168.200.X'
    option 'netmask' '255.255.255.0'

#### WAN configuration
config 'interface' 'wan'
    option 'ifname' 'eth0.2'
    option 'proto' 'dhcp'

#### Wi-Fi LAN configuration
config 'interface' 'wifi'
    option 'ifname' 'wlan0'
    option 'proto' 'static'
    option 'ipaddr' '192.168.3.X'
    option 'netmask' '255.255.255.0'
```

Otro cambio importante será la configuración del fichero */etc/config/wireless*, que nos permitirá establecer algunos parámetros de configuración para la interfaz inalámbrica; se ha decidido usar el canal 5 en todos los routers.

D.9 Modificación del Fichero wireless

X será el identificador asignado al router, empezando a contar en 1

```
config 'wifi-device' 'wlan0'
    option 'type' 'mac80211'
    option 'channel' '5'
    option 'disabled' '0'

config 'wifi-iface'
    option 'device' 'wlan0'
    option 'mode' 'adhoc'
    option 'ssid' 'linksys-X'
    option 'encryption' 'none'
```

Sería recomendable asignar un nombre único a cada router; de este modo, sabremos en todo momento qué router se está configurando. Esto se consigue editando el fichero */etc/config/system* y asignando un nombre, siguiendo la pauta *linksysX* (donde *X* será el identificador numérico del router).

Es imprescindible editar el fichero */etc/sysctl.conf*; de no hacerlo, el router en cuestión no reenviará los paquetes que le lleguen cuyo destino no sea él mismo.

D.10 Habilitar el Reenvío de Paquetes en los Routers

Editar el fichero /etc/sysctl.conf

Descomentar la siguiente linea

```
# net.ipv6.conf.all.forwarding=1
```

El mecanismo de acceso a los routers mediante SSH, por defecto, solicita una contraseña cada vez que se intenta acceder. Esto es un problema a la hora de ejecutar scripts automatizados, pues el script se detendrá a la espera de que se introduzca la contraseña solicitada. Por este motivo, se ha de modificar este comportamiento y conseguir que el password no deba ser introducido

manualmente, sino que los routers compartan con el PC una clave pública, que le permita a este último acceder a los routers sin necesidad de introducir la mencionada contraseña.

D.11 Habilitar el Acceso a SSH sin password

Crear una clave publica en el PC
ssh-keygen -t dsa
Copiar dicha clave a los routers
scp ~/.ssh/id_dsa.pub IPdelRouter:/tmp/
Crear las claves de autorización en los routers
cd /etc/dropbear/
cat /tmp/id*.pub >> authorized_keys
chmod 0600 authorized_keys

APENDICE E - Instalación y Configuración del Software Quagga y olsrd en el Entorno Openwrt

Tras terminar la fase de instalación del firmware y su posterior configuración, los routers ya están preparados para instalar el software de Quagga.

E.1 Instalar Quagga en los Routers

scp *.ipk root@IPdelRouter:/tmp/

ssh root@IPdelRouter

opkg install /tmp/quagga_0.99.9-1_mipsel.ipk

opkg install /tmp/quagga-libzebra_0.99.9-1_mipsel.ipk

opkg install /tmp/quagga-libospf_0.99.9-1_mipsel.ipk

opkg install /tmp/quagga-ospfd_0.99.9-1_mipsel.ipk

opkg install /tmp/quagga-ospf6d_0.99.9-1_mipsel.ipk

Una vez instalado, únicamente quedará copiar y ajustar ciertos parámetros en los ficheros de configuración *ospf6d.conf, osfpd.conf* y *zebra.conf* (apéndice E.2), antes de poder arrancarlo (apéndice E.3).
La configuración de los demonios puede realizarse mediante línea de comandos o editando directamente los ficheros de configuración con cualquier editor de textos. En nuestro caso, los parámetros utilizados serán los recomendados por el desarrollador del parche de MANET.

E.2 Copiar los Ficheros de Configuración

scp *.conf root@IPdelRouter:/etc/quagga/

Contenido de los ficheros de configuración:

ospf6d.conf

```
! -*- ospf -*-
!
! OSPFd sample configuration file
!
!
hostname ospf6d
password zebra
log file /tmp/ospf6d.log debugging
!log stdout
!
debug ospf6 lsa unknown
!
interface wlan0
 ipv6 ospf6 cost 1
 ipv6 ospf6 hello-interval 10
```

```
 ipv6 ospf6 dead-interval 40
 ipv6 ospf6 retransmit-interval 5
 ipv6 ospf6 priority 1
 ipv6 ospf6 transmit-delay 1
 ipv6 ospf6 instance-id 0
 ipv6 ospf6 network point-to-multipoint
 ipv6 ospf6 flood-delay 100
!
router ospf6
 router-id 192.168.1.1
 redistribute connected
 redistribute ospf
 interface wlan0 area 192.168.0.0
!
line vty
!
```

zebra.conf

```
! -*- zebra -*-
!
! zebra sample configuration file
!
! $Id: zebra.conf.sample,v 1.1.1.1 2002/12/13 20:15:30 paul Exp $
!
hostname Router
password zebra
enable password zebra
!
! Interface's description.
!
!interface lo
! description test of desc.
!
!interface sit0
! multicast

!
! Static default route sample.
!
!ip route 0.0.0.0/0 203.181.89.241
!

!log file zebra.log
```

E.3 Arrancar los Demonios de Quagga

```
/etc/init.d/quagga start
```

E.4 Acceder a la Interfaces de Configuración de los Demonios

Zebra

```
telnet 127.0.0.1 2601
```

Ospf6

```
telnet 127.0.0.1 2606
```

Para la instalación y configuración de OLSR hay que seguir los siguientes pasos:

E.5 Instalar olsrd en los Routers

```
scp *.ipk root@IPdelRouter:/tmp/

ssh root@IPdelRouter

opkg install /tmp/olsrd.ipk
```

Una vez instalado [OLSRDIPV6], únicamente quedará ajustar ciertos parámetros en el fichero de configuración olsrd.conf

E.6 Fichero de Configuración olsrd

```
DebugLevel 0
IpVersion 6
AllowNoInt yes
Pollrate 0.025
TcRedundancy 2
MprCoverage 3
LinkQualityFishEye 1
LinkQualityWinSize 100
LinkQualityDijkstraLimit 0 9.0
LinkQualityLevel 2
UseHysteresis no
FIBMetric "flat"
ClearScreen yes
Willingness 3
LinkQualityAging 0.1
LinkQualityAlgorithm "etx_fpm"

Interface "wlan0"
{
   Ip4Broadcast 255.255.255.255
   HelloInterval 2.0
   HelloValidityTime 40.0
   TcInterval 5.0
   TcValidityTime 100.0
   MidInterval 18.0
   MidValidityTime 324.0
   HnaInterval 18.0
   HnaValidityTime 108.0
}
```

Una vez correctamente configurado solo queda arrancarlo mediante su comando correspondiente.

E.7 Arrancar Demonio olsrd

```
./olsrd
```

APENDICE F – Scripts

F.1 config.sh

```
#!/bin/bash
#config.sh

echo "***** CONFIGURACIÓN COMPLETA *****"
echo "***** ROUTER 1 *****"
ssh root@192.168.2.1 "ip -6 addr add 3ffe:10::1/128 dev wlan0;ip6tables -P INPUT
DROP;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:1a:64 -j
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:90:4c:5f:00:2a -j
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:32:d0 -j
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:32:3a -j
ACCEPT;/etc/init.d/quagga start"
echo "***** ROUTER 2 *****"
ssh root@192.168.2.2 "ip -6 addr add 3ffe:20::1/128 dev wlan0;ip6tables -P INPUT
DROP;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:32:ca -j
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:35:19 -j
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:88:92:40 -j
ACCEPT;/etc/init.d/quagga start"
echo "***** ROUTER 3 *****"
ssh root@192.168.2.3 "ip -6 addr add 3ffe:3::1/128 dev wlan0;ip6tables -P INPUT
DROP;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:32:ca -j
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:35:19 -j
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:3b:be -j
ACCEPT;/etc/init.d/quagga start"
echo "***** ROUTER 4 *****"
ssh root@192.168.2.4 "ip -6 addr add 3ffe:4::1/128 dev wlan0;ip6tables -P INPUT
DROP;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:32:ca -j
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:1a:4f -j
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:3b:be -j
ACCEPT;/etc/init.d/quagga start"
echo "***** ROUTER 5 *****"
ssh root@192.168.2.5 "ip -6 addr add 3ffe:5::1/128 dev wlan0;ip6tables -P INPUT
DROP;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:32:ca -j
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:88:92:40 -j
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:1a:4f -j
ACCEPT;/etc/init.d/quagga start"
echo "***** ROUTER 6 *****"
ssh root@192.168.2.6 "ip -6 addr add 3ffe:6::1/128 dev wlan0;ip6tables -P INPUT
DROP;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:1a:64 -j
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:90:4c:5f:00:2a -j
ACCEPT;/etc/init.d/quagga start"
echo "***** ROUTER 7 *****"
ssh root@192.168.2.7 "ip -6 addr add 3ffe:7::1/128 dev wlan0;ip6tables -P INPUT
DROP;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:1a:64 -j
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:32:3a -j
ACCEPT;/etc/init.d/quagga start"
echo "***** ROUTER 8 *****"
ssh root@192.168.2.8 "ip -6 addr add 3ffe:8::1/128 dev wlan0;ip6tables -P INPUT
DROP;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:32:d0 -j
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:32:3a -j
ACCEPT;/etc/init.d/quagga start"
echo "***** ROUTER 9 *****"
ssh root@192.168.2.9 "ip -6 addr add 3ffe:9::1/128 dev wlan0;ip6tables -P INPUT
DROP;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:90:4c:5f:00:2a -j
```

```
ACCEPT;ip6tables -A INPUT -i wlan0 -m mac --mac-source 00:1c:10:44:32:d0 -j
ACCEPT;/etc/init.d/quagga start"
echo "***** CONFIGURACION CARGADA *****"
sleep 20

./quagga.sh

sleep 20
# Se prueba la comunicacion entre R8 y R6 y se tirara R4 que previamente
# hemos comprobado que interviene en el camino entre ambos.
# Almacenamos la informacion (tarda cerca de 34 segundos)
echo "ping"
ssh root@192.168.2.8 "ping6 3ffe:6::1 -c 50 > /etc/ficheroping &"
sleep 5
ssh root@192.168.2.8 "cat /etc/ficheroping"
sleep 5
ssh root@192.168.2.8 "cat /etc/ficheroping"
echo "ifdown"
ssh root@192.168.2.4 "ifdown wifi"
sleep 45
echo "copia"
scp root@192.168.2.8:/etc/ficheroping .
# Todo en una sola linea
cat ficheroping | grep seq | awk '{ print $5 }' | awk -F "=" '{ print $2 }' >
numPrueba

# Analisis de la informacion

# Cargamos el fichero con los tiempos
miarray=(`cat numPrueba`)
echo " la longitud es " ${#miarray[*]}
# Aqui guardaremos el valor anterior para compararlo con el siguiente
let "anterior=0"
# Establecemos el valor maximo de la diferencia que queramos
# para considerar que se cae el enlace y se recupera
let "diferencia=5"
# Recorremos el array
for num in "${miarray[@]}"
do
   # Si es el primer elemento no hacemos nada
   if [ $anterior != 0 ]
   then
      # Vemos la resta para comprobar que no son consecutivos
      let "resta=$num-$anterior"
      if [ $resta -gt $diferencia ]
      then
         echo "El tiempo de recuperación ha sido de " $resta
      fi
   fi
   anterior=$num
done
```

F.2 quagga.sh

```
#!/bin/bash
# quagga.sh
# Arranca quagga en todos los routes
```

```
# ESTE SCRIPT SOLO FUNCIONA PARA 9 NODOS

echo "***** ARRANCANDO QUAGGA *****"
echo "***** ROUTER 1 *****"
ssh root@192.168.2.1 "/etc/init.d/quagga start"
echo "***** ROUTER 2 *****"
ssh root@192.168.2.2 "/etc/init.d/quagga start"
echo "***** ROUTER 3 *****"
ssh root@192.168.2.3 "/etc/init.d/quagga start"
echo "***** ROUTER 4 *****"
ssh root@192.168.2.4 "/etc/init.d/quagga start"
echo "***** ROUTER 5 *****"
ssh root@192.168.2.5 "/etc/init.d/quagga start"
echo "***** ROUTER 6 *****"
ssh root@192.168.2.6 "/etc/init.d/quagga start"
echo "***** ROUTER 7 *****"
ssh root@192.168.2.7 "/etc/init.d/quagga start"
echo "***** ROUTER 8 *****"
ssh root@192.168.2.8 "/etc/init.d/quagga start"
echo "***** ROUTER 9 *****"
ssh root@192.168.2.9 "/etc/init.d/quagga start"
echo "***** QUAGGA INICIALIZADO *****"
```

F.3 reinciar.sh

```
#!/bin/bash
# reiniciar.sh
# Reinicia todos los routes
# ESTE SCRIPT SOLO FUNCIONA PARA 9 NODOS

echo "***** ROUTER 9 Reiniciado *****"
ssh root@192.168.2.9 "reboot"
echo "***** ROUTER 8 Reiniciado *****"
ssh root@192.168.2.8 "reboot"
echo "***** ROUTER 7 Reiniciado *****"
ssh root@192.168.2.7 "reboot"
echo "***** ROUTER 6 Reiniciado *****"
ssh root@192.168.2.6 "reboot"
echo "***** ROUTER 5 Reiniciado *****"
ssh root@192.168.2.5 "reboot"
echo "***** ROUTER 4 Reiniciado *****"
ssh root@192.168.2.4 "reboot"
echo "***** ROUTER 3 Reiniciado *****"
ssh root@192.168.2.3 "reboot"
echo "***** ROUTER 2 Reiniciado *****"
ssh root@192.168.2.2 "reboot"
echo "***** ROUTER 1 Reiniciado *****"
ssh root@192.168.2.1 "reboot"
echo "***** 9 ROUTERS REINICIADOS *****"
```

F.4 generadorTopologias.sh

```
#!/bin/bash

#
#               X R6
#
```

```
#          X R3        X R2
#
#     X R9        X R1        X R7
#
#          X R4        X R5
#
#               X R8
#
# Genera automaticamente una topologia en base a la informacion
# pasada en el fichero nodos
# Radio de alcance fijo (por parametro) o variable (por fichero)
# mediante el parametro -r seguido del radio (entre 0 y 500)

# Declaracion de variables

ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)

if [ $# -eq 0 ]
then
   radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":" '{ print $2 }'`)
else
   if [ $# -eq 2 ]
   then
      if [ $1 = "-r" ]
      then
         if [ $2 -gt 0 ]
         then
            if [ $2 -lt 500 ]
            then
               let "cont_radios=0"
               for i in "${ips[@]}"
               do
                  radios[$cont_radios]=$2
                  let "cont_radios++"
               done
            else
               echo "El radio introducido es superior al permitido (500)"
               exit
            fi
         else
            echo "El radio introducido no es valido, introduzca un numero positivo"
            exit
         fi
      else
         echo "La opcion $1 no es valida, use -r"
         exit
      fi
   else
      echo "Script generador de topologias wireless"
      echo "El fichero con la informacion de los equipos ha de llamarse \"nodos\""
      echo "Funcionamiento:"
      echo "Sin parametros -> Lee toda la informacion del fichero nodos"
      echo "-r metros      -> Establece un radio de cobertura igual para todos los
equipos"
```

```
        exit
    fi
fi

# Contadores de referencia para los routers: 0=R1, 1=R2, etc ...

let "i=0"
let "j=0"

# 0 = Rechazar la conexion, 1 = Aceptar la conexion

# Fin de variables

for x in "${coordX[@]}"
do
    for x2 in "${coordX[@]}"
    do
        if [ $i -ne $j ]
        then
            c1=`echo "($x-$x2)^2" | bc`
            c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
            h=`echo "scale=0; sqrt($c1+$c2)" | bc`
            let "dist_max=${radios[$i]}+${radios[$j]}"
            if [ $h -lt $dist_max ]
            then
                R[$j]=1
            else
                R[$j]=0
            fi
        else
            R[$i]=0
        fi
        let "j++"
    done
    # Aqui ya tenemos R lleno para i
    let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
    ipv6R="3ffe:$numR::1"
    echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]} -"
    let "c=0"
    comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev wlan0;ip6tables
-P INPUT DROP\""
    eval "$comando"
    for valor in "${R[@]}"
    do
        if [ $valor -eq "1" ]
        then
            comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${macs[$c]} -j ACCEPT\""
            let "numC=($c+1)"
            echo "R$numC esta dentro de alcance"
            eval "$comando"
        fi
        let "c++"
    done
    comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
    eval "$comando"
    let "v1=0"
```

```
    for v in "${R[@]}"
    do
        R[$v1]=0
        let "v1++"
    done
    let "j=0"
    let "i++"
done
echo "Configuracion finalizada correctamente"
```

F.5 Fichero de Entrada "nodos"

```
192.168.2.1 00:1c:10:44:32:ca x:150 y:150 r:100
192.168.2.2 00:1c:10:44:1a:64 x:150 y:300 r:100
192.168.2.3 00:90:4c:5f:00:2a x:0 y:150 r:100
192.168.2.4 00:1c:10:44:32:d0 x:150 y:0 r:100
192.168.2.5 00:1c:10:44:32:3a x:300 y:150 r:100
192.168.2.6 00:1c:10:44:35:19 x:0 y:300 r:100
192.168.2.7 00:1c:10:88:92:40 x:300 y:300 r:100
192.168.2.8 00:1c:10:44:1a:4f x:300 y:0 r:100
192.168.2.9 00:1c:10:44:3b:be x:0 y:0 r:100
```

F.6 generadorTopologias.sh

```
#!/bin/bash

#
#              X R6
#
#         X R3       X R2
#
#    X R9       X R1       X R7
#
#         X R4       X R5
#
#              X R8
#
# Radio de alcance variable para cada router o fijo
# mediante el parametro -r seguido del radio (entre 0 y 500)
# Los nodos extremos del enlace a tirar se pasan mediante
# -n1 ipNodo1 - n2 ipNodo2

# Declaracion de variables

ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)

if [ $# -eq 5 ]
then
    if [ $5 = "-r" ]
    then
        if [ $6 -gt 0 ]
        then
            if [ $6 -lt 500 ]
            then
```

```
            let "cont_radios=0"
            for i in "${ips[@]}"
            do
               radios[$cont_radios]=$6
               let "cont_radios++"
            done
            if [ $1 = "-n1" ]
            then
               nodo1=$2
               let "c1=0"
               for x in "${ips[@]}"
               do
                  if [ $x = $nodo1 ]
                  then
                     nodoMAC1=${macs[$c1]}
                     echo "$nodoMAC1"
                  fi
                  let "c1++"
               done
            fi
            if [ $3 = "-n2" ]
            then
               nodo2=$4
               let "c1=0"
               for x in "${ips[@]}"
               do
                  if [ $x = $nodo2 ]
                  then
                     nodoMAC2=${macs[$c1]}
                     echo "$nodoMAC2"
                  fi
                  let "c1++"
               done
            fi
         else
            echo "El radio introducido es superior al permitido (500)"
            exit
         fi
      else
         echo "El radio introducido no es valido, introduzca un numero positivo"
         exit
      fi
   else
      echo "La opcion $6 no es valida, use -r"
      exit
   fi
else
   if [ $# -eq 4 ]
   then
      if [ $1 = "-n1" ]
      then
         radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":" '{ print $2
}'`)
         nodo1=$2
         let "c1=0"
         for x in "${ips[@]}"
         do
```

```
            if [ $x = $nodo1 ]
            then
               nodoMAC1=${macs[$c1]}
               echo "$nodoMAC1"
            fi
            let "c1++"
         done
      fi
      if [ $3 = "-n2" ]
      then
         nodo2=$4
         let "c1=0"
         for x in "${ips[@]}"
         do
            if [ $x = $nodo2 ]
            then
               nodoMAC2=${macs[$c1]}
               echo "$nodoMAC2"
            fi
            let "c1++"
         done
      fi
   else
      echo "Script generador de topologias wireless que deshabilita un enlace"
      echo "El fichero con la informacion de los equipos ha de llamarse \"nodos\""
      echo "Funcionamiento:"
      echo "tiraenlaces -n1 ipNodo1 -n2 ipNodo2"
      echo "Deshabilita el enlace entre ambos nodos cogiendo los radios del fichero
nodos"
      echo "tiraenlaces -n1 ipNodo1 -n2 ipNodo2 -r radio"
      echo "Igual que el anterior pero el radio es establecido como \"radio\" para
todos los nodos"
      exit
   fi
fi

# Contadores de referencia para los routers: 0=R1, 1=R2, etc ...

let "vueltas=10"
let "numvuelta=0"
let "i=0"
let "j=0"

# 0 = Rechazar la conexion, 1 = Aceptar la conexion

# Fin de variables

./reiniciar.sh
echo "Esperando 90 seg. para que los routers esten listos"
sleep 90

for x in "${coordX[@]}"
do
   for x2 in "${coordX[@]}"
   do
      if [ $i -ne $j ]
      then
```

```
            c1=`echo "($x-$x2)^2" | bc`
            c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
            h=`echo "scale=0; sqrt($c1+$c2)" | bc`
            let "dist_max=${radios[$i]}+${radios[$j]}"
            if [ $h -lt $dist_max ]
            then
               R[$j]=1
            else
               R[$j]=0
            fi
         else
            R[$i]=0
         fi
         let "j++"
      done

      # Aqui ya tenemos R lleno para i
      let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
      ipv6R="3ffe:$numR::1"
      echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]} -"
      let "c=0"
      comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev wlan0;ip6tables
-P INPUT DROP\""
      eval "$comando"
      for valor in "${R[@]}"
      do
         if [ $valor -eq "1" ]
         then
            comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${macs[$c]} -j ACCEPT\""
            let "numC=($c+1)"
            echo "R$numC esta dentro de alcance"
            eval "$comando"
         fi
         let "c++"
      done
      comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
      eval "$comando"

      # Ponemos a 0 el array
      let "v1=0"
      for v in "${R[@]}"
      do
         R[$v1]=0
         let "v1++"
      done

      let "j=0"
      let "i++"
done

# Esperamos un tiempo y arrancamos los quagga para asegurarnos

# sleep 20
# ./quagga.sh
# sleep 20
```

```
# Aqui tiramos el nodo elejido y hacemos ping de todos con todos

let "i=1"
let "j=1"

while [ $numvuelta -lt $vueltas ]
do
   echo "Vuelta $numvuelta"
   for x in "${coordX[@]}"
   do
      for x2 in "${coordX[@]}"
      do
         if [ $i -ne $j ]
         then
            # Rearrancamos quagga para asegurar que esta activo
            ./quagga.sh 1>>_log_tiraenlaces
            sleep 5
            # ----------------------------------------
            # hacemos ping de i a j
            # tiramos el nodo
            # calculamos el tiempo de convergencia
            # Almacenamos la informacion (tarda cerca de 34 segundos)
            echo "----------------------------------------"
            echo "Haciendo ping desde 3ffe:$i::1 hacia 3ffe:$j::1, esperando 10
seg."
            echo "----------------------------------------"
            ssh root@192.168.2.$i "ping6 3ffe:$j::1 -c 50 > /etc/_temp_ficheroping
&"
            sleep 5
            # ssh root@192.168.2.$i "cat /etc/_temp_ficheroping"
            sleep 5
            # ssh root@192.168.2.$i "cat /etc/_temp_ficheroping"
            # TIRAMOS EL ENLACE del nodo seleccionado
            # Nodo 1
            echo "MAC $nodoMAC2 rechazada para $nodo1"
            ssh root@$nodo1 "ip6tables -D INPUT -i wlan0 -m mac --mac-source
$nodoMAC2 -j ACCEPT"
            # Nodo 2
            echo "MAC $nodoMAC1 rechazada para $nodo2, esperando 45 seg"
            ssh root@$nodo2 "ip6tables -D INPUT -i wlan0 -m mac --mac-source
$nodoMAC1 -j ACCEPT"
            # Estaba a 50
            sleep 45
            echo "Copiando informacion del ping en local: _temp_ficheroping"
            scp root@192.168.2.$i:/etc/_temp_ficheroping . 1>>_log_tiraenlaces
            # Todo en una sola linea
            cat _temp_ficheroping | grep seq | awk '{ print $5 }' | awk -F "=" '{
print $2 }' > _temp_numPrueba

            # Analisis de la informacion

            # Cargamos el fichero con los tiempos
            miarray=(`cat _temp_numPrueba`)
            echo " Nº de echo reply recibidos: ${#miarray[*]}"
            # Aqui guardaremos el valor anterior para compararlo con el siguiente
            let "anterior=0"
            # Establecemos el valor maximo de la diferencia que queramos
```

```
            # para considerar que se cae el enlace y se recupera
            let "diferencia=5"
            # Recorremos el array
            for num in "${miarray[@]}"
            do
               # Si es el primer elemento no hacemos nada
               if [ $anterior != 0 ]
               then
                  # Vemos la resta para comprobar que no son consecutivos
                  let "resta=$num-$anterior"
                  if [ $resta -gt $diferencia ]
                  then
                     echo "--> El tiempo de recuperación ha sido de $resta"
                     echo $resta >> _tiempos_tiraenlaces
                  fi
               fi
               anterior=$num
            done
            # ----------------------------------------
            # Levantamos de nuevo el enlace para el siguiente
            # Levantamos el nodo
            # Nodo 1
            echo "MAC $nodoMAC2 aceptada para $nodo1"
            ssh root@$nodo1 "ip6tables -A INPUT -i wlan0 -m mac --mac-source
$nodoMAC2 -j ACCEPT"
            # Nodo 2
            echo "MAC $nodoMAC1 aceptada para $nodo2"
            ssh root@$nodo2 "ip6tables -A INPUT -i wlan0 -m mac --mac-source
$nodoMAC1 -j ACCEPT"
            echo "Esperamos 25 seg."
            sleep 25
         fi
         let "j++"
      done
      let "j=1"
      let "i++"
   done
   let "i=1"
   let "numvuelta++"
done # fin del while

# Ahora tenemos que leer el fichero tiempos para hacer la media

# Cargamos el fichero con los tiempos
miarray=(`cat _tiempos_tiraenlaces`)
let "numtiempos=${#miarray[*]}"
let "media=0"
# Aqui guardaremos el valor anterior para compararlo con el siguiente
let "anterior=0"
# Establecemos el valor maximo de la diferencia que queramos
# para considerar que se cae el enlace y se recupera
let "diferencia=5"
# Recorremos el array
for num in "${miarray[@]}"
do
   let "media+=$num"
done
```

```
let "media/=$numtiempos"
echo "El tiempo de convergencia medio es $media"
```

F.7 tiraenlaces.sh

```
#!/bin/bash

#
#             X R6
#
#         X R3       X R2
#
#    X R9       X R1       X R7
#
#         X R4       X R5
#
#             X R8
#
# Radio de alcance variable para cada router o fijo
# mediante el parametro -r seguido del radio (entre 0 y 500)
# Los nodos extremos del enlace a tirar se pasan mediante
# -n1 ipNodo1 - n2 ipNodo2

# Declaracion de variables

ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)

if [ $# -eq 5 ]
then
        if [ $5 = "-r" ]
        then
                if [ $6 -gt 0 ]
                then
                        if [ $6 -lt 500 ]
                        then
                                let "cont_radios=0"
                                for i in "${ips[@]}"
                                do
                                        radios[$cont_radios]=$6
                                        let "cont_radios++"
                                done
                                if [ $1 = "-n1" ]
                                then
                                        nodo1=$2
                                        let "c1=0"
                                        for x in "${ips[@]}"
                                        do
                                                if [ $x = $nodo1 ]
                                                then
                                                        nodoMAC1=${macs[$c1]}
                                                        echo "$nodoMAC1"
                                                fi
                                                let "c1++"
                                        done
```

```
                        fi
                        if [ $3 = "-n2" ]
                        then
                                nodo2=$4
                                let "c1=0"
                                for x in "${ips[@]}"
                                do
                                        if [ $x = $nodo2 ]
                                        then
                                                nodoMAC2=${macs[$c1]}
                                                echo "$nodoMAC2"
                                        fi
                                        let "c1++"
                                done
                        fi
                else
                        echo "El radio introducido es superior al permitido
(500)"
                        exit
                fi
        else
                echo "El radio introducido no es valido, introduzca un numero
positivo"
                exit
        fi
    else
        echo "La opcion $6 no es valida, use -r"
        exit
    fi
else
    if [ $# -eq 4 ]
    then
        if [ $1 = "-n1" ]
        then
                radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":"
'{ print $2 }'`)
                nodo1=$2
                let "c1=0"
                for x in "${ips[@]}"
                do
                        if [ $x = $nodo1 ]
                        then
                                nodoMAC1=${macs[$c1]}
                                echo "$nodoMAC1"
                        fi
                        let "c1++"
                done
        fi
        if [ $3 = "-n2" ]
        then
                nodo2=$4
                let "c1=0"
                for x in "${ips[@]}"
                do
                        if [ $x = $nodo2 ]
                        then
                                nodoMAC2=${macs[$c1]}
```

```
                                        echo "$nodoMAC2"
                                fi
                                let "c1++"
                        done
                fi
        else
                echo "Script generador de topologias wireless que deshabilita un
enlace"
                echo "El fichero con la informacion de los equipos ha de llamarse
\"nodos\""
                echo "Funcionamiento:"
                echo "tiraenlaces -n1 ipNodo1 -n2 ipNodo2"
                echo "Deshabilita el enlace entre ambos nodos cogiendo los radios del
fichero nodos"
                echo "tiraenlaces -n1 ipNodo1 -n2 ipNodo2 -r radio"
                echo "Igual que el anterior pero el radio es establecido como \"radio\"
para todos los nodos"
                exit
        fi
fi

# Contadores de referencia para los routers: 0=R1, 1=R2, etc ...

let "vueltas=10"
let "numvuelta=0"
let "i=0"
let "j=0"

# 0 = Rechazar la conexion, 1 = Aceptar la conexion

# Fin de variables

./reiniciar.sh
echo "Esperando 90 seg. para que los routers esten listos"
sleep 90

for x in "${coordX[@]}"
do
        for x2 in "${coordX[@]}"
        do
                if [ $i -ne $j ]
                then
                        c1=`echo "($x-$x2)^2" | bc`
                        c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
                        h=`echo "scale=0; sqrt($c1+$c2)" | bc`
                        let "dist_max=${radios[$i]}+${radios[$j]}"
                        if [ $h -lt $dist_max ]
                        then
                                R[$j]=1
                        else
                                R[$j]=0
                        fi
                else
                        R[$i]=0
                fi
                let "j++"
        done
```

```
        # Aqui ya tenemos R lleno para i
        let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
        ipv6R="3ffe:$numR::1"
        echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]}
-"
        let "c=0"
        comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev
wlan0;ip6tables -P INPUT DROP\""
        eval "$comando"
        for valor in "${R[@]}"
        do
                if [ $valor -eq "1" ]
                then
                        comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0
-m mac --mac-source ${macs[$c]} -j ACCEPT\""
                        let "numC=($c+1)"
                        echo "R$numC esta dentro de alcance"
                        eval "$comando"
                fi
                let "c++"
        done
        comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
        eval "$comando"

        # Ponemos a 0 el array
        let "v1=0"
        for v in "${R[@]}"
        do
                R[$v1]=0
                let "v1++"
        done

        let "j=0"
        let "i++"
done

# Esperamos un tiempo y arrancamos los quagga para asegurarnos

# sleep 20
# ./quagga.sh
# sleep 20

# Aqui tiramos el nodo elejido y hacemos ping de todos con todos

let "i=1"
let "j=1"

while [ $numvuelta -lt $vueltas ]
do
        echo "Vuelta $numvuelta"
        for x in "${coordX[@]}"
        do
                for x2 in "${coordX[@]}"
                do
                        if [ $i -ne $j ]
                        then
```

```
                        # Rearrancamos quagga para asegurar que esta activo
                        ./quagga.sh 1>>_log_tiraenlaces
                        sleep 5
                        # ----------------------------------------
                        # hacemos ping de i a j
                        # tiramos el nodo
                        # calculamos el tiempo de convergencia
                        # Almacenamos la informacion (tarda cerca de 34
segundos)
                        echo "----------------------------------------"
                        echo "Haciendo ping desde 3ffe:$i::1 hacia 3ffe:$j::1,
esperando 10 seg."
                        echo "----------------------------------------"
                        ssh root@192.168.2.$i "ping6 3ffe:$j::1 -c 50 >
/etc/_temp_ficheroping &"
                        sleep 5
                        # ssh root@192.168.2.$i "cat /etc/_temp_ficheroping"
                        sleep 5
                        # ssh root@192.168.2.$i "cat /etc/_temp_ficheroping"
                        # TIRAMOS EL ENLACE del nodo seleccionado
                        # Nodo 1
                        echo "MAC $nodoMAC2 rechazada para $nodo1"
                        ssh root@$nodo1 "ip6tables -D INPUT -i wlan0 -m mac
--mac-source $nodoMAC2 -j ACCEPT"
                        # Nodo 2
                        echo "MAC $nodoMAC1 rechazada para $nodo2,
esperando 45 seg"
                        ssh root@$nodo2 "ip6tables -D INPUT -i wlan0 -m mac
--mac-source $nodoMAC1 -j ACCEPT"
                        # Estaba a 50
                        sleep 45
                        echo "Copiando informacion del ping en local:
_temp_ficheroping"
                        scp root@192.168.2.$i:/etc/_temp_ficheroping .
1>>_log_tiraenlaces
                        # Todo en una sola linea
                        cat _temp_ficheroping | grep seq | awk '{ print $5 }' |
awk -F "=" '{ print $2 }' > _temp_numPrueba

                        # Analisis de la informacion

                        # Cargamos el fichero con los tiempos
                        miarray=(`cat _temp_numPrueba`)
                        echo " NÂº de echo reply recibidos: ${#miarray[*]}"
                        # Aqui guardaremos el valor anterior para compararlo
con el siguiente
                        let "anterior=0"
                        # Establecemos el valor maximo de la diferencia que
queramos
                        # para considerar que se cae el enlace y se recupera
                        let "diferencia=5"
                        # Recorremos el array
                        for num in "${miarray[@]}"
                        do
                                # Si es el primer elemento no hacemos nada
                                if [ $anterior != 0 ]
                                then
```

```
                                        # Vemos la resta para comprobar que no
son consecutivos
                                        let "resta=$num-$anterior"
                                        if [ $resta -gt $diferencia ]
                                        then
                                                echo "--> El tiempo de
recuperaciÃ³n ha sido de $resta"
                                                echo $resta >>
_tiempos_tiraenlaces
                                        fi
                                fi
                                anterior=$num
                        done
                        # ----------------------------------------
                        # Levantamos de nuevo el enlace para el siguiente
                        # Levantamos el nodo
                        # Nodo 1
                        echo "MAC $nodoMAC2 aceptada para $nodo1"
                        ssh root@$nodo1 "ip6tables -A INPUT -i wlan0 -m mac
--mac-source $nodoMAC2 -j ACCEPT"
                        # Nodo 2
                        echo "MAC $nodoMAC1 aceptada para $nodo2"
                        ssh root@$nodo2 "ip6tables -A INPUT -i wlan0 -m mac
--mac-source $nodoMAC1 -j ACCEPT"
                        echo "Esperamos 25 seg."
                        sleep 25
                fi
                let "j++"
        done
        let "j=1"
        let "i++"
    done
    let "i=1"
    let "numvuelta++"
done # fin del while

# Ahora tenemos que leer el fichero tiempos para hacer la media

# Cargamos el fichero con los tiempos
miarray=(`cat _tiempos_tiraenlaces`)
let "numtiempos=${#miarray[*]}"
let "media=0"
# Aqui guardaremos el valor anterior para compararlo con el siguiente
let "anterior=0"
# Establecemos el valor maximo de la diferencia que queramos
# para considerar que se cae el enlace y se recupera
let "diferencia=5"
# Recorremos el array
for num in "${miarray[@]}"
do
    let "media+=$num"
done
let "media/=$numtiempos"
echo "El tiempo de convergencia medio es $media"
```

F.8 tiempo_arranque.sh

```
#!/bin/bash

# Declaracion de variables

ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)
radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":" '{ print $2 }'`)

let "vueltas=20"
let "numvuelta=0"

while [ $numvuelta -lt $vueltas ]
do
   echo "Vuelta $numvuelta"

let "i=0"
let "j=0"

./reiniciar.sh
echo "Esperando 90 seg. para que los routers esten listos"
sleep 90

tiempoInicio=`date +%s`

for x in "${coordX[@]}"
do
   for x2 in "${coordX[@]}"
   do
      if [ $i -ne $j ]
      then
         c1=`echo "($x-$x2)^2" | bc`
         c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
         h=`echo "scale=0; sqrt($c1+$c2)" | bc`
         let "dist_max=${radios[$i]}+${radios[$j]}"
         if [ $h -lt $dist_max ]
         then
            R[$j]=1
         else
            R[$j]=0
         fi
      else
         R[$i]=0
      fi
      let "j++"
   done

   # Aqui ya tenemos R lleno para i
   let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
   ipv6R="3ffe:$numR::1"
   echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]} -"
   let "c=0"
   comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev wlan0;ip6tables
-P INPUT DROP\""
   eval "$comando"
   for valor in "${R[@]}"
```

```
    do
        if [ $valor -eq "1" ]
        then
            comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${macs[$c]} -j ACCEPT\""
            let "numC=($c+1)"
            echo "R$numC esta dentro de alcance"
            eval "$comando"
        fi
        let "c++"
    done

    # Ponemos a 0 el array
    let "v1=0"
    for v in "${R[@]}"
    do
        R[$v1]=0
        let "v1++"
    done

    let "j=0"
    let "i++"
done

let "i=0"
for x2 in "${coordX[@]}"
do
    comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
    eval "$comando" >>_log_tiempo_arranque
    let "i++"
done

let "e=0"
fin="false"
while [ $fin != "true" ]
do
    sleep 1
    let "i=0"

    # LANZO EL IP ROUTE PARA VER QUE TODOS LOS NODOS SON ACCESIBLES
    comando="ssh root@${ips[$i]} \"ip -6 route\""
    eval "$comando" > _tiempo_arranque_iproute
    rutas=(`cat _tiempo_arranque_iproute | grep '3ffe:' | awk '{ print $1 }'`)
    if [ ${#rutas[@]} == ${#ips[@]} ]
    then
        tiempoFin=`date +%s`
        let "tiempoTotal=$tiempoFin-$tiempoInicio"
        fin="true"
    else
        for x2 in "${coordX[@]}"
        do
            comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
            eval "$comando" >>_log_tiempo_arranque
            let "i++"
        done
        let "e++"
    fi
```

```
done

echo "$tiempoTotal" >> _tiempos_arranque
echo "El tiempo de arranque ha sido: $tiempoTotal"

let "numvuelta++"

done # fin del while
```

F.9 tiempo_nodo_nuevo.sh

```
#!/bin/bash

# Declaracion de variables

ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)
radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":" '{ print $2 }'`)

let "vueltas=20"
let "numvuelta=0"

let "nodoAunir=${#ips[@]}-1"

while [ $numvuelta -lt $vueltas ]
do
   echo "Vuelta $numvuelta"

let "i=0"
let "j=0"

./reiniciar.sh
echo "Esperando 90 seg. para que los routers esten listos"
sleep 90

for x in "${coordX[@]}"
do
   for x2 in "${coordX[@]}"
   do
      if [ $i -ne $j ]
      then
         c1=`echo "($x-$x2)^2" | bc`
         c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
         h=`echo "scale=0; sqrt($c1+$c2)" | bc`
         let "dist_max=${radios[$i]}+${radios[$j]}"
         if [ $h -lt $dist_max ]
         then
            R[$j]=1
         else
            R[$j]=0
         fi
      else
         R[$i]=0
      fi
      let "j++"
```

```
    done

    # Aqui ya tenemos R lleno para i
    let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
    ipv6R="3ffe:$numR::1"
    echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]} -"
    let "c=0"
    comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev wlan0;ip6tables
-P INPUT DROP\""
    eval "$comando"
    for valor in "${R[@]}"
    do
        if [ $valor -eq "1" ]
        then
            comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${macs[$c]} -j ACCEPT\""
            let "numC=($c+1)"
            echo "R$numC esta dentro de alcance"
            eval "$comando"
        fi
        let "c++"
    done

    # Ponemos a 0 el array
    let "v1=0"
    for v in "${R[@]}"
    do
        R[$v1]=0
        let "v1++"
    done

    let "j=0"
    let "i++"
done

let "i=0"
for x2 in "${coordX[@]}"
do
    if [ $nodoAunir != $i ]
    then
        comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
        eval "$comando" >>_log_tiempo_nodo_nuevo
    fi
    let "i++"
done

# ESPERAMOS A QUE SE ESTABILICEN LAS RUTAS
let "e=0"
fin="false"
while [ $fin != "true" ]
do
    sleep 1
    let "i=0"

    # LANZO EL IP ROUTE PARA VER QUE TODOS LOS NODOS SON ACCESIBLES
    comando="ssh root@${ips[$i]} \"ip -6 route\""
    eval "$comando" > _tiempo_arranque_iproute
```

```
   rutas=(`cat _tiempo_arranque_iproute | grep '3ffe:' | awk '{ print $1 }'`)
   if [ ${#rutas[@]} == $nodoAunir ]
   then
      fin="true"
   else
      for x2 in "${coordX[@]}"
      do
         if [ $nodoAunir != $i ]
         then
            comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
            eval "$comando" >>_log_tiempo_nodo_nuevo
         fi
         let "i++"
      done
      let "e++"
   fi
done

# INTRODUCIMOS EL NODO NUEVO Y COMPROBAMOS CUANDO SE ESTABILIZA LA
RED
comando="ssh root@${ips[$nodoAunir]} \"/etc/init.d/quagga start\""
eval "$comando" >>_log_tiempo_nodo_nuevo

tiempoInicio=`date +%s`

let "e=0"
fin="false"
while [ $fin != "true" ]
do
   sleep 1
   let "i=0"

   # LANZO EL IP ROUTE PARA VER QUE TODOS LOS NODOS SON ACCESIBLES
   comando="ssh root@${ips[$i]} \"ip -6 route\""
   eval "$comando" > _tiempo_arranque_iproute
   rutas=(`cat _tiempo_arranque_iproute | grep '3ffe:' | awk '{ print $1 }'`)
   if [ ${#rutas[@]} == ${#ips[@]} ]
   then
      tiempoFin=`date +%s`
      let "tiempoTotal=$tiempoFin-$tiempoInicio"
      fin="true"
   else
      for x2 in "${coordX[@]}"
      do
            comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
            eval "$comando" >>_log_tiempo_nodo_nuevo
         let "i++"
      done
      let "e++"
   fi
done

# Medimos el tiempo de inicio
# Tiramos la interfaz wifi del nodo
# Levantamos el nodo y lo configuramos como antes

echo "$tiempoTotal" >> _tiempos_nodo_nuevo
```

```
echo "El tiempo de arranque ha sido: $tiempoTotal"

let "numvuelta++"

done # fin del while
```

F.10 carga_sin_errores_total.sh

```
#!/bin/bash

# Declaracion de variables

ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)
radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":" '{ print $2 }'`)

let "vueltas=2"
let "numvuelta=0"

./reiniciar.sh
echo "Esperando 90 seg. para que los routers esten listos"
sleep 90

sudo ip link set wlan0 down
sudo iwconfig wlan0 mode Ad-hoc channel 5
sudo ip link set wlan0 up
sudo ip ad add 3ffe:100::1/128 dev wlan0
echo "Estado de wlan0:"
echo "----------------"
iwconfig
echo "----------------"

let "i=0"
let "j=0"

for x in "${coordX[@]}"
do
   for x2 in "${coordX[@]}"
   do
      if [ $i -ne $j ]
      then
         c1=`echo "($x-$x2)^2" | bc`
         c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
         h=`echo "scale=0; sqrt($c1+$c2)" | bc`
         let "dist_max=${radios[$i]}+${radios[$j]}"
         if [ $h -lt $dist_max ]
         then
            R[$j]=1
         else
            R[$j]=0
         fi
      else
         R[$i]=0
      fi
      let "j++"
```

```
   done

   # Aqui ya tenemos R lleno para i
   let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
   ipv6R="3ffe:$numR::1"
   echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]} -"
   let "c=0"
   comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev wlan0;ip6tables
-P INPUT DROP\""
   eval "$comando"
   for valor in "${R[@]}"
   do
      if [ $valor -eq "1" ]
      then
         comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${macs[$c]} -j ACCEPT\""
         let "numC=($c+1)"
         echo "R$numC esta dentro de alcance"
         eval "$comando"
      fi
      let "c++"
   done
   comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
   eval "$comando"

   # Ponemos a 0 el array
   let "v1=0"
   for v in "${R[@]}"
   do
      R[$v1]=0
      let "v1++"
   done

   let "j=0"
   let "i++"
done

# CODIGO NUEVO
sleep 30
let "i=0"
for x2 in "${coordX[@]}"
do
   comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
   eval "$comando"
   let "i++"
done
sleep 30
let "i=0"
for x2 in "${coordX[@]}"
do
   comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
   eval "$comando"
   let "i++"
done
sleep 30
let "i=0"
for x2 in "${coordX[@]}"
```

```
do
   comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
   eval "$comando"
   let "i++"
done
# CODIGO NUEVO

let "link=0"

for lnk in "${macs[@]}"
do

   while [ $numvuelta -lt $vueltas ]
   do
      echo "Capturando trafico Prueba $link de ${ips[$i]} Vuelta ($numvuelta de
$vueltas)"

      sudo tshark -i wlan0 -f "proto ospf" -a duration:60 -w /tmp/captura_tshark &

      let "e=0"

      while [ $e -lt "4" ]
      do
         sleep 5
         let "i=0"
         for x2 in "${coordX[@]}"
         do
            comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
            eval "$comando"
            let "i++"
         done
         let "e++"
      done

      capinfos /tmp/captura_tshark >> _carga_sin_errores_Total
      echo "Datos almacenados en el fichero _carga_sin_errores_Total"

      sleep 5

      let "numvuelta++"

   done # fin del while

   echo "***************** FIN PRUEBA NODO ${macs[$link]}
*******************" >> _carga_sin_errores_Total
   let "numvuelta=0"

   let "link++"

done
```

F.11 carga_sin_errores_por_nodo.sh

```
#!/bin/bash

# Declaracion de variables
```

```
ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)
radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":" '{ print $2 }'`)

let "numpruebas=0"
let "pruebas=3"

let "vueltas=20"
let "numvuelta=0"

./reiniciar.sh
echo "Esperando 90 seg. para que los routers esten listos"
sleep 90

sudo ip link set wlan0 down
sudo iwconfig wlan0 mode Ad-hoc channel 5
sudo ip link set wlan0 up
sudo ip ad add 3ffe:100::1/128 dev wlan0
echo "Estado de wlan0:"
echo "----------------"
iwconfig
echo "----------------"

let "i=0"
let "j=0"

for x in "${coordX[@]}"
do
   for x2 in "${coordX[@]}"
   do
      if [ $i -ne $j ]
      then
         c1=`echo "($x-$x2)^2" | bc`
         c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
         h=`echo "scale=0; sqrt($c1+$c2)" | bc`
         let "dist_max=${radios[$i]}+${radios[$j]}"
         if [ $h -lt $dist_max ]
         then
            R[$j]=1
         else
            R[$j]=0
         fi
      else
         R[$i]=0
      fi
      let "j++"
   done

   # Aqui ya tenemos R lleno para i
   let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
   ipv6R="3ffe:$numR::1"
   echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]} -"
   let "c=0"
   comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev wlan0;ip6tables
-P INPUT DROP\""
```

```
    eval "$comando"
    for valor in "${R[@]}"
    do
        if [ $valor -eq "1" ]
        then
            comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${macs[$c]} -j ACCEPT\""
            let "numC=($c+1)"
            echo "R$numC esta dentro de alcance"
            eval "$comando"
        fi
        let "c++"
    done
    comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
    eval "$comando"

    # Ponemos a 0 el array
    let "v1=0"
    for v in "${R[@]}"
    do
        R[$v1]=0
        let "v1++"
    done

    let "j=0"
    let "i++"
done

# CODIGO NUEVO
sleep 30
let "i=0"
for x2 in "${coordX[@]}"
do
    comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
    eval "$comando"
    let "i++"
done
sleep 30
let "i=0"
for x2 in "${coordX[@]}"
do
    comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
    eval "$comando"
    let "i++"
done
sleep 30
let "i=0"
for x2 in "${coordX[@]}"
do
    comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
    eval "$comando"
    let "i++"
done
# CODIGO NUEVO

while [ $numpruebas -lt $pruebas ]
do
```

```
   let "link=0"

   for lnk in "${macs[@]}"
   do

      while [ $numvuelta -lt $vueltas ]
      do
         echo "Capturando trafico del nodo $link con MAC ${macs[$link]} Vuelta
($numvuelta de $vueltas)"

         sudo tshark -i wlan0 -f "proto ospf and ether src ${macs[$link]}" -a
duration:60 -w /tmp/captura_tshark &

         let "e=0"

         while [ $e -lt "3" ]
         do
            sleep 5
            let "i=0"
            for x2 in "${coordX[@]}"
            do
               comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
               eval "$comando"
               let "i++"
            done
            let "e++"
         done

         capinfos /tmp/captura_tshark >> _carga_sin_errores_por_Nodo
         echo "Datos almacenados en el fichero _carga_sin_errores_por_Nodo"

         sleep 5

         let "numvuelta++"

      done # fin del while

      echo "***************** FIN PRUEBA NODO ${macs[$link]}
********************" >> _carga_sin_errores_por_Nodo
      let "numvuelta=0"

      let "link++"

   done

   let "numpruebas++"

done
```

F.12 pico_carga_con_erroresv2.sh

```
#!/bin/bash

# BORRAR EL FICHERO _pico_carga_enlaces PARA CADA PRUEBA

# SOLO REINICIA AL PRINCIPIO, SE USA ESTA
```

```
# Script encargado de obtener todos los enlaces de la topología para realizar
# las pruebas de picos de carga. Se tira cada uno de los enlaces "$vueltas" veces
# para comprobar los picos de carga producidos en cada uno de los enlaces.

# Declaracion de variables

ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)
radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":" '{ print $2 }'`)

let "vueltas=20"
let "numvuelta=0"

./reiniciar.sh
echo "Esperando 90 seg. para que los routers esten listos"
sleep 90

sudo ip link set wlan0 down
sudo iwconfig wlan0 mode Ad-hoc channel 5
sudo ip link set wlan0 up
sudo ip ad add 3ffe:100::1/128 dev wlan0
echo "Estado de wlan0:"
echo "----------------"
iwconfig
echo "----------------"

let "i=0"
let "j=0"

for x in "${coordX[@]}"
do
   for x2 in "${coordX[@]}"
   do
      if [ $i -ne $j ]
      then
         c1=`echo "($x-$x2)^2" | bc`
         c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
         h=`echo "scale=0; sqrt($c1+$c2)" | bc`
         let "dist_max=${radios[$i]}+${radios[$j]}"
         if [ $h -lt $dist_max ]
         then
            R[$j]=1
         else
            R[$j]=0
         fi
      else
         R[$i]=0
      fi
      let "j++"
   done

   # Aqui ya tenemos R lleno para i
   let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
   ipv6R="3ffe:$numR::1"
```

```
   echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]} -"
   let "c=0"
   comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev wlan0;ip6tables
-P INPUT DROP\""
   eval "$comando"
   for valor in "${R[@]}"
   do
      if [ $valor -eq "1" ]
      then
         comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${macs[$c]} -j ACCEPT\""
         let "numC=($c+1)"
         echo "R$numC esta dentro de alcance"
         eval "$comando"
         # GUARDAMOS CADA UNO DE LOS ENLACES EN UN FICHERO PARA HACER
LUEGO LA PRUEBA
         if [ $i -lt $c ]
         then
            echo "${ips[$i]}-${macs[$c]}-${ips[$c]}-${macs[$i]}" >>
_pico_carga_enlaces
         fi
      fi
      let "c++"
   done
   comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
   eval "$comando"

   # Ponemos a 0 el array
   let "v1=0"
   for v in "${R[@]}"
   do
      R[$v1]=0
      let "v1++"
   done

   let "j=0"
   let "i++"
done

# AQUI YA LO TENEMOS LLENO PROCESAMOS EL FICHERO EN ARRAYS PARA IR
TIRANDO LOS ENLACES
# ENLACE A -> B
enlace_ips_N1=(`cat _pico_carga_enlaces | awk -F "-" '{ print $1 }'`)
enlace_mac_N2=(`cat _pico_carga_enlaces | awk -F "-" '{ print $2 }'`)
# ENLACE B -> A
enlace_ips_N2=(`cat _pico_carga_enlaces | awk -F "-" '{ print $3 }'`)
enlace_mac_N1=(`cat _pico_carga_enlaces | awk -F "-" '{ print $4 }'`)

# CODIGO NUEVO
sleep 30
let "i=0"
for x2 in "${coordX[@]}"
do
   comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
   eval "$comando"
   let "i++"
done
```

```
sleep 30
let "i=0"
for x2 in "${coordX[@]}"
do
   comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
   eval "$comando"
   let "i++"
done
sleep 30
let "i=0"
for x2 in "${coordX[@]}"
do
   comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
   eval "$comando"
   let "i++"
done
# CODIGO NUEVO

let "link=0"

for lnk in "${enlace_ips_N1[@]}"
do

   while [ $numvuelta -lt $vueltas ]
   do
      echo "Enlace ($link de ${#enlace_ips_N1[@]}) Vuelta ($numvuelta de
$vueltas)"

      sudo tshark -i wlan0 -f "proto ospf" -a duration:60 -w /tmp/captura_tshark &

      # AQUI TENEMOS QUE IR TIRANDO EL ENLACE 1 A 1

      # TIRAMOS EL ENLACE
      echo "MAC ${enlace_mac_N2[$link]} rechazada para
${enlace_ips_N1[$link]}"
      ssh root@${enlace_ips_N1[$link]} "ip6tables -D INPUT -i wlan0 -m mac --
mac-source ${enlace_mac_N2[$link]} -j ACCEPT"
      echo "MAC ${enlace_mac_N1[$link]} rechazada para
${enlace_ips_N2[$link]}"
      ssh root@${enlace_ips_N2[$link]} "ip6tables -D INPUT -i wlan0 -m mac --
mac-source ${enlace_mac_N1[$link]} -j ACCEPT"

      let "e=0"

      while [ $e -lt "4" ]
      do
         sleep 5
         let "i=0"
         for x2 in "${coordX[@]}"
         do
            comando="ssh root@${ips[$i]} \"/etc/init.d/quagga start\""
            eval "$comando"
            let "i++"
         done
         let "e++"
      done
```

```
        capinfos /tmp/captura_tshark >> _pico_carga_con_errores_E
        echo "Datos almacenados en el fichero _pico_carga_con_errores_E"

        # LEVANTAMOS EL ENLACE
        echo "MAC ${enlace_mac_N2[$link]} aceptada para
${enlace_ips_N1[$link]}"
        ssh root@${enlace_ips_N1[$link]} "ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${enlace_mac_N2[$link]} -j ACCEPT"
        echo "MAC ${enlace_mac_N1[$link]} aceptada para
${enlace_ips_N2[$link]}"
        ssh root@${enlace_ips_N2[$link]} "ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${enlace_mac_N1[$link]} -j ACCEPT"

        sleep 15

        let "numvuelta++"

    done # fin del while

    echo "FIN PRUEBA ENLACE ${enlace_ips_N1[$link]} - ${enlace_ips_N2[$link]}"
>> _pico_carga_con_errores_E
    let "numvuelta=0"

    let "link++"

done
```

F.13 generaNodos.sh

```
#!/bin/bash

# Script encargado de generar una topologia aleatoria bidimensional.
# Parte de las coordenadas X e Y ( ambas en 0 ) y mediante un numero
# aleatorio que sirve para seleccionar si se aumenta X o Y va generando
# nuevos nodos garantizando la conectividad entre todos ellos.

let "nodos=20"
let "actual=2"
let "X=500"
let "Y=500"
let "radio=150"
let "r=100"
let "cero=0"
existeX="false"
existeY="false"

macs=(`cat macs`)

BINARY=2
T=1

rm nodos
echo "192.168.2.1 ${macs[0]} x:500 y:500 r:$r" >> nodos

while [ $actual -le $nodos ]
do
    number=$RANDOM
```

```
   numberSigno=$RANDOM
   let "indice=$actual-1"
   let "number %= $BINARY"
   let "numberSigno %= $BINARY"
   if [ "$number" -eq $T ]
   then
      if [ "$numberSigno" -eq $T ]
      then
         let "comprobar=$X-$radio"
         if [ $comprobar -ge $cero ]
         then
            coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print
$2 }'`)
            coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print
$2 }'`)
            for x in "${coordX[@]}"
            do
            if [ $x -eq $comprobar ]
            then
               existeX="true"
            fi
            done
            for y in "${coordY[@]}"
            do
            if [ $y -eq $Y ]
            then
               existeY="true"
            fi
            done
            if [ $existeX = "true" ] && [ $existeY = "true" ]
            then
               let "cero=0"
            else
               let "X -= $radio"
               echo "192.168.2.$actual ${macs[$indice]} x:$X y:$Y r:$r" >> nodos
               let "actual++"
            fi
         else
            let "numsuma=$X+100"
            coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print
$2 }'`)
            coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print
$2 }'`)
            for x in "${coordX[@]}"
            do
            if [ $x -eq $numsuma ]
            then
               existeX="true"
            fi
            done
            for y in "${coordY[@]}"
            do
            if [ $y -eq $Y ]
            then
               existeY="true"
            fi
            done
```

```
            if [ $existeX = "true" ] && [ $existeY = "true" ]
            then
               let "cero=0"
            else
               let "X += $radio"
               echo "192.168.2.$actual ${macs[$indice]} x:$X y:$Y r:$r" >> nodos
               let "actual++"
            fi
         fi
      else
            let "numsuma=$X+100"
            coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print
$2 }'`)
            coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print
$2 }'`)
            for x in "${coordX[@]}"
            do
            if [ $x -eq $numsuma ]
            then
               existeX="true"
            fi
            done
            for y in "${coordY[@]}"
            do
            if [ $y -eq $Y ]
            then
               existeY="true"
            fi
            done
            if [ $existeX = "true" ] && [ $existeY = "true" ]
            then
               let "cero=0"
            else
               let "X += $radio"
               echo "192.168.2.$actual ${macs[$indice]} x:$X y:$Y r:$r" >> nodos
               let "actual++"
            fi
      fi
      existeX="false"
      existeY="false"
   else
      if [ "$numberSigno" -eq $T ]
      then
         let "comprobar=$Y-$radio"
         if [ $comprobar -ge $cero ]
         then
            coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print
$2 }'`)
            coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print
$2 }'`)
            for y in "${coordY[@]}"
            do
            if [ $y -eq $comprobar ]
            then
               existeY="true"
            fi
            done
```

```
            for x in "${coordX[@]}"
            do
            if [ $x -eq $X ]
            then
               existeX="true"
            fi
            done
            if [ $existeX = "true" ] && [ $existeY = "true" ]
            then
               let "cero=0"
            else
               let "Y -= $radio"
               echo "192.168.2.$actual ${macs[$indice]} x:$X y:$Y r:$r" >> nodos
               let "actual++"
            fi
         else
            let "numsuma=$Y+100"
            coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print
$2 }'`)
            coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print
$2 }'`)
            for y in "${coordY[@]}"
            do
            if [ $y -eq $numsuma ]
            then
               existeY="true"
            fi
            done
            for x in "${coordX[@]}"
            do
            if [ $x -eq $X ]
            then
               existeX="true"
            fi
            done
            if [ $existeX = "true" ] && [ $existeY = "true" ]
            then
               let "cero=0"
            else
               let "Y += $radio"
               echo "192.168.2.$actual ${macs[$indice]} x:$X y:$Y r:$r" >> nodos
               let "actual++"
            fi
         fi
      else
            let "numsuma=$Y+100"
            coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print
$2 }'`)
            coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print
$2 }'`)
            for y in "${coordY[@]}"
            do
            if [ $y -eq $numsuma ]
            then
               existeY="true"
            fi
            done
```

```
            for x in "${coordX[@]}"
            do
            if [ $x -eq $X ]
            then
               existeX="true"
            fi
            done
            if [ $existeX = "true" ] && [ $existeY = "true" ]
            then
               let "cero=0"
            else
               let "Y += $radio"
               echo "192.168.2.$actual ${macs[$indice]} x:$X y:$Y r:$r" >> nodos
               let "actual++"
            fi
      fi
   fi
   existeX="false"
   existeY="false"
done

echo "Topologia almacenada en el fichero \"nodos\""
cat nodos
```

F.15 tiranodos.sh

```
#!/bin/bash

#
#              X R6
#
#         X R3       X R2
#
#    X R9       X R1       X R7
#
#         X R4       X R5
#
#              X R8
#
# Diseñado para 9 routers
# Radio de alcance variable para cada router o fijo
# mediante el parametro -r seguido del radio (entre 0 y 500)

# Declaracion de variables

ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)

if [ $# -eq 4 ]
then
   if [ $1 = "-n" ]
   then
      nodo=$2
   else
```

```
        echo "La opcion $1 no es valida, use -n ipNodo"
        exit
    fi
    if [ $3 = "-r" ]
    then
        if [ $4 -gt 0 ]
        then
            if [ $4 -lt 500 ]
            then
                let "cont_radios=0"
                for i in "${ips[@]}"
                do
                    radios[$cont_radios]=$4
                    let "cont_radios++"
                done
            else
                echo "El radio introducido es superior al permitido (500)"
                exit
            fi
        else
            echo "El radio introducido no es valido, introduzca un numero positivo"
            exit
        fi
    else
        echo "La opcion $3 no es valida, use -r"
        exit
    fi
else
    if [ $# -eq 2 ]
    then
        if [ $1 = "-n" ]
        then
            radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":" '{ print $2
}'`)
            nodo=$2
        else
            echo "La opcion $1 no es valida, use -n ipNodo"
            exit
        fi
    else
        echo "Script generador de topologias wireless que deshabilita un nodo"
        echo "El fichero con la informacion de los equipos ha de llamarse \"nodos\""
        echo "Funcionamiento:"
        echo "-n ipNodo          -> Deshabilita el nodo en cuestion durante las
pruebas"
        echo "-n ipNodo -r radio  -> Igual que el anterior pero con radio fijo \"radio\""
        exit
    fi
fi

# Contadores de referencia para los routers: 0=R1, 1=R2, etc ...

# nodo="192.168.2.3"
let "vueltas=10"
let "numvuelta=0"
let "i=0"
let "j=0"
```

```
# 0 = Rechazar la conexion, 1 = Aceptar la conexion

# Fin de variables

./reiniciar.sh
sleep 65

for x in "${coordX[@]}"
do
   for x2 in "${coordX[@]}"
   do
      if [ $i -ne $j ]
      then
         c1=`echo "($x-$x2)^2" | bc`
         c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
         h=`echo "scale=0; sqrt($c1+$c2)" | bc`
         let "dist_max=${radios[$i]}+${radios[$j]}"
         if [ $h -lt $dist_max ]
         then
            R[$j]=1
         else
            R[$j]=0
         fi
      else
         R[$i]=0
      fi
      let "j++"
   done

   # Aqui ya tenemos R lleno para i
   let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
   ipv6R="3ffe:$numR::1"
   echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]} -"
   let "c=0"
   comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev wlan0;ip6tables
-P INPUT DROP\""
   eval "$comando"
   for valor in "${R[@]}"
   do
      if [ $valor -eq "1" ]
      then
         comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${macs[$c]} -j ACCEPT\""
         let "numC=($c+1)"
         echo "R$numC esta dentro de alcance"
         eval "$comando"
      fi
      let "c++"
   done
   comando="ssh root@${ips[$i]} \"olsrd\""
   eval "$comando"

   # Ponemos a 0 el array
   let "v1=0"
   for v in "${R[@]}"
   do
```

```
        R[$v1]=0
        let "v1++"
    done

    let "j=0"
    let "i++"
done

# Aqui tiramos el nodo elejido y hacemos ping de todos con todos

let "i=1"
let "j=1"

sleep 20

while [ $numvuelta -lt $vueltas ]
do
    echo "Vuelta $numvuelta"
    for x in "${coordX[@]}"
    do
        for x2 in "${coordX[@]}"
        do
            if [ $i -ne $j ]
            then
                # ------------------------------------------
                # hacemos ping de i a j
                # tiramos el nodo
                # calculamos el tiempo de convergencia
                # Almacenamos la informacion (tarda cerca de 34 segundos)
                echo "Haciendo ping desde 3ffe:$i::1 hacia 3ffe:$j::1"
                ssh root@192.168.2.$i "ping6 3ffe:$j::1 -c 50 > /etc/_temp_ficheroping
&"
                sleep 5
                # Tiramos el nodo
                echo "Nodo $nodo tirado, esperando 45 seg."
                ssh root@$nodo "ifdown wifi"
                sleep 45
                echo "Copiando informacion del ping en local: _temp_ficheroping"
                scp root@192.168.2.$i:/etc/_temp_ficheroping . 1>>_log_tiranodos
                # Todo en una sola linea
                cat _temp_ficheroping | grep seq | awk '{ print $5 }' | awk -F "=" '{
print $2 }' > _temp_numPrueba

                # Analisis de la informacion

                # Cargamos el fichero con los tiempos
                miarray=(`cat _temp_numPrueba`)
                echo " Nº de echo reply recibidos: ${#miarray[*]}"
                # Aqui guardaremos el valor anterior para compararlo con el siguiente
                let "anterior=0"
                # Establecemos el valor maximo de la diferencia que queramos
                # para considerar que se cae el enlace y se recupera
                let "diferencia=5"
                # Recorremos el array
                for num in "${miarray[@]}"
                do
                    # Si es el primer elemento no hacemos nada
```

```
                if [ $anterior != 0 ]
                then
                    # Vemos la resta para comprobar que no son consecutivos
                    let "resta=$num-$anterior"
                    if [ $resta -gt $diferencia ]
                    then
                        echo "El tiempo de recuperación ha sido de $resta"
                        echo $resta >> _tiempos_tiranodos
                    fi
                fi
                anterior=$num
            done
            # ----------------------------------------
            # Levantamos de nuevo el enlace para el siguiente
            # Levantamos el nodo
            echo "Levantando el nodo $nodo, esperando 60 seg."
            ssh root@$nodo "ifup wifi"
            numero=(`echo $nodo | awk -F "." '{ print $4 }'`)
            ssh root@$nodo "ip addr add 3ffe:$numero::1/128 dev wlan0"
            sleep 40
        fi
        let "j++"
    done
    let "j=1"
    let "i++"
  done
  let "i=1"
  let "numvuelta++"
done # fin del while

# Ahora tenemos que leer el fichero tiempos para hacer la media

# Cargamos el fichero con los tiempos
miarray=(`cat _tiempos_tiranodos`)
let "numtiempos=${#miarray[*]}"
let "media=0"
# Aqui guardaremos el valor anterior para compararlo con el siguiente
let "anterior=0"
# Establecemos el valor maximo de la diferencia que queramos
# para considerar que se cae el enlace y se recupera
let "diferencia=5"
# Recorremos el array
for num in "${miarray[@]}"
do
  let "media+=$num"
done
let "media/=$numtiempos"
echo "El tiempo de convergencia medio es $media"
```

F.16 tiraenlaces.sh

```
#!/bin/bash

#
#              X R6
#
#         X R3       X R2
```

```
#
#    X R9       X R1       X R7
#
#         X R4       X R5
#
#              X R8
#
# Hace ping de todos con todos y va tirando un enlace para comprobar
# como afecta a la topologia entera y cuanto tarda en recuperarse en
# los casos en los que interviniese dicho enlace.
# Radio de alcance variable para cada router o fijo
# mediante el parametro -r seguido del radio (entre 0 y 500)
# Los nodos extremos del enlace a tirar se pasan mediante
# -n1 ipNodo1 - n2 ipNodo2

# Declaracion de variables

ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)

if [ $# -eq 5 ]
then
   if [ $5 = "-r" ]
   then
      if [ $6 -gt 0 ]
      then
         if [ $6 -lt 500 ]
         then
            let "cont_radios=0"
            for i in "${ips[@]}"
            do
               radios[$cont_radios]=$6
               let "cont_radios++"
            done
            if [ $1 = "-n1" ]
            then
               nodo1=$2
               let "c1=0"
               for x in "${ips[@]}"
               do
                  if [ $x = $nodo1 ]
                  then
                     nodoMAC1=${macs[$c1]}
                     echo "$nodoMAC1"
                  fi
                  let "c1++"
               done
            fi
            if [ $3 = "-n2" ]
            then
               nodo2=$4
               let "c1=0"
               for x in "${ips[@]}"
               do
                  if [ $x = $nodo2 ]
```

```
                then
                    nodoMAC2=${macs[$c1]}
                    echo "$nodoMAC2"
                fi
                let "c1++"
            done
        fi
    else
        echo "El radio introducido es superior al permitido (500)"
        exit
    fi
  else
    echo "El radio introducido no es valido, introduzca un numero positivo"
    exit
  fi
else
  if [ $# -eq 4 ]
  then
    if [ $1 = "-n1" ]
    then
      radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":" '{ print $2
}'`)
      nodo1=$2
      let "c1=0"
      for x in "${ips[@]}"
      do
        if [ $x = $nodo1 ]
        then
          nodoMAC1=${macs[$c1]}
          echo "$nodoMAC1"
        fi
        let "c1++"
      done
    fi
    if [ $3 = "-n2" ]
    then
      nodo2=$4
      let "c1=0"
      for x in "${ips[@]}"
      do
        if [ $x = $nodo2 ]
        then
          nodoMAC2=${macs[$c1]}
          echo "$nodoMAC2"
        fi
        let "c1++"
      done
    fi
  else
    echo "Script generador de topologias wireless que deshabilita un enlace"
    echo "El fichero con la informacion de los equipos ha de llamarse \"nodos\""
    echo "Funcionamiento:"
    echo "tiraenlaces -n1 ipNodo1 -n2 ipNodo2"
```

```
        echo "Deshabilita el enlace entre ambos nodos cogiendo los radios del fichero
nodos"
        echo "tiraenlaces -n1 ipNodo1 -n2 ipNodo2 -r radio"
        echo "Igual que el anterior pero el radio es establecido como \"radio\" para
todos los nodos"
        exit
    fi
fi

# Contadores de referencia para los routers: 0=R1, 1=R2, etc ...

let "vueltas=10"
let "numvuelta=0"
let "i=0"
let "j=0"

# 0 = Rechazar la conexion, 1 = Aceptar la conexion

# Fin de variables

./reiniciar.sh
echo "Esperando 65 seg. para que los routers esten listos"
sleep 65

for x in "${coordX[@]}"
do
    for x2 in "${coordX[@]}"
    do
        if [ $i -ne $j ]
        then
            c1=`echo "($x-$x2)^2" | bc`
            c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
            h=`echo "scale=0; sqrt($c1+$c2)" | bc`
            let "dist_max=${radios[$i]}+${radios[$j]}"
            if [ $h -lt $dist_max ]
            then
                R[$j]=1
            else
                R[$j]=0
            fi
        else
            R[$i]=0
        fi
        let "j++"
    done

    # Aqui ya tenemos R lleno para i
    let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
    ipv6R="3ffe:$numR::1"
    echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]} -"
    let "c=0"
    comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev wlan0;ip6tables
-P INPUT DROP\""
    eval "$comando"
    for valor in "${R[@]}"
    do
        if [ $valor -eq "1" ]
```

```
        then
            comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${macs[$c]} -j ACCEPT\""
            let "numC=($c+1)"
            echo "R$numC esta dentro de alcance"
            eval "$comando"
        fi
        let "c++"
    done
    comando="ssh root@${ips[$i]} \"olsrd\""
    eval "$comando"

    # Ponemos a 0 el array
    let "v1=0"
    for v in "${R[@]}"
    do
        R[$v1]=0
        let "v1++"
    done

    let "j=0"
    let "i++"
done

let "i=1"
let "j=1"

sleep 20

while [ $numvuelta -lt $vueltas ]
do
    echo "Vuelta $numvuelta"
    for x in "${coordX[@]}"
    do
        for x2 in "${coordX[@]}"
        do
            if [ $i -ne $j ]
            then
                # ----------------------------------------
                # hacemos ping de i a j
                # tiramos el nodo
                # calculamos el tiempo de convergencia
                # Almacenamos la informacion (tarda cerca de 34 segundos)
                echo "----------------------------------------"
                echo "Haciendo ping desde 3ffe:$i::1 hacia 3ffe:$j::1, esperando 5 seg."
                echo "----------------------------------------"
                ssh root@192.168.2.$i "ping6 3ffe:$j::1 -c 50 > /etc/_temp_ficheroping
&"
                sleep 5
                # TIRAMOS EL ENLACE del nodo seleccionado
                # Nodo 1
                echo "MAC $nodoMAC2 rechazada para $nodo1"
                ssh root@$nodo1 "ip6tables -D INPUT -i wlan0 -m mac --mac-source
$nodoMAC2 -j ACCEPT"
                # Nodo 2
                echo "MAC $nodoMAC1 rechazada para $nodo2, esperando 45 seg"
                ssh root@$nodo2 "ip6tables -D INPUT -i wlan0 -m mac --mac-source
```

```
$nodoMAC1 -j ACCEPT"
            # Estaba a 50
            sleep 45
            echo "Copiando informacion del ping en local: _temp_ficheroping"
            scp root@192.168.2.$i:/etc/_temp_ficheroping . 1>>_log_tiraenlaces
            # Todo en una sola linea
            cat _temp_ficheroping | grep seq | awk '{ print $5 }' | awk -F "=" '{
print $2 }' > _temp_numPrueba

            # Analisis de la informacion

            # Cargamos el fichero con los tiempos
            miarray=(`cat _temp_numPrueba`)
            echo " Nº de echo reply recibidos: ${#miarray[*]}"
            # Aqui guardaremos el valor anterior para compararlo con el siguiente
            let "anterior=0"
            # Establecemos el valor maximo de la diferencia que queramos
            # para considerar que se cae el enlace y se recupera
            let "diferencia=5"
            # Recorremos el array
            for num in "${miarray[@]}"
            do
               # Si es el primer elemento no hacemos nada
               if [ $anterior != 0 ]
               then
                  # Vemos la resta para comprobar que no son consecutivos
                  let "resta=$num-$anterior"
                  if [ $resta -gt $diferencia ]
                  then
                     echo "--> El tiempo de recuperación ha sido de $resta"
                     echo $resta >> _tiempos_tiraenlaces
                  fi
               fi
               anterior=$num
            done
            # ----------------------------------------
            # Levantamos de nuevo el enlace para el siguiente
            # Levantamos el nodo
            # Nodo 1
            echo "MAC $nodoMAC2 aceptada para $nodo1"
            ssh root@$nodo1 "ip6tables -A INPUT -i wlan0 -m mac --mac-source
$nodoMAC2 -j ACCEPT"
            # Nodo 2
            echo "MAC $nodoMAC1 aceptada para $nodo2"
            ssh root@$nodo2 "ip6tables -A INPUT -i wlan0 -m mac --mac-source
$nodoMAC1 -j ACCEPT"
            echo "Esperamos 25 seg."
            sleep 25
         fi
         let "j++"
      done
      let "j=1"
      let "i++"
   done
   let "i=1"
   let "numvuelta++"
done # fin del while
```

```
# Ahora tenemos que leer el fichero tiempos para hacer la media

# Cargamos el fichero con los tiempos
miarray=(`cat _tiempos_tiraenlaces`)
let "numtiempos=${#miarray[*]}"
let "media=0"
# Aqui guardaremos el valor anterior para compararlo con el siguiente
let "anterior=0"
# Establecemos el valor maximo de la diferencia que queramos
# para considerar que se cae el enlace y se recupera
let "diferencia=5"
# Recorremos el array
for num in "${miarray[@]}"
do
   let "media+=$num"
done
let "media/=$numtiempos"
echo "El tiempo de convergencia medio es $media"
```

F.17 tiempo_arranque.sh

```
#!/bin/bash

# Declaracion de variables

ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)
radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":" '{ print $2 }'`)

let "vueltas=20"
let "numvuelta=0"

while [ $numvuelta -lt $vueltas ]
do
   echo "Vuelta $numvuelta"

let "i=0"
let "j=0"

./reiniciar.sh
echo "Esperando 65 seg. para que los routers esten listos"
sleep 65

tiempoInicio=`date +%s`

for x in "${coordX[@]}"
do
   for x2 in "${coordX[@]}"
   do
      if [ $i -ne $j ]
      then
         c1=`echo "($x-$x2)^2" | bc`
         c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
         h=`echo "scale=0; sqrt($c1+$c2)" | bc`
```

```
            let "dist_max=${radios[$i]}+${radios[$j]}"
            if [ $h -lt $dist_max ]
            then
               R[$j]=1
            else
               R[$j]=0
            fi
         else
            R[$i]=0
         fi
         let "j++"
      done

      # Aqui ya tenemos R lleno para i
      let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
      ipv6R="3ffe:$numR::1"
      echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]} -"
      let "c=0"
      comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev wlan0;ip6tables
-P INPUT DROP\""
      eval "$comando"
      for valor in "${R[@]}"
      do
         if [ $valor -eq "1" ]
         then
            comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${macs[$c]} -j ACCEPT\""
            let "numC=($c+1)"
            echo "R$numC esta dentro de alcance"
            eval "$comando"
         fi
         let "c++"
      done

      # Ponemos a 0 el array
      let "v1=0"
      for v in "${R[@]}"
      do
         R[$v1]=0
         let "v1++"
      done

      let "j=0"
      let "i++"
done

let "i=0"
for x2 in "${coordX[@]}"
do
      comando="ssh root@${ips[$i]} \"olsrd\""
      eval "$comando" >>_log_tiempo_arranque
      let "i++"
done

sleep 20

fin="false"
```

```
while [ $fin != "true" ]
do
   sleep 1
   let "i=0"

   # LANZO EL IP ROUTE PARA VER QUE TODOS LOS NODOS SON ACCESIBLES
   comando="ssh root@${ips[$i]} \"ip -6 route\""
   eval "$comando" > _tiempo_arranque_iproute
   rutas=(`cat _tiempo_arranque_iproute | grep '3ffe:' | awk '{ print $1 }'`)
   if [ ${#rutas[@]} == ${#ips[@]} ]
   then
      tiempoFin=`date +%s`
      let "tiempoTotal=$tiempoFin-$tiempoInicio"
      fin="true"
   fi
done

echo "$tiempoTotal" >> _tiempos_arranque
echo "El tiempo de arranque ha sido: $tiempoTotal"

let "numvuelta++"

done # fin del while
```

F.18 tiempo_nodo_nuevo.sh

```
#!/bin/bash

# Declaracion de variables

ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)
radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":" '{ print $2 }'`)

let "vueltas=20"
let "numvuelta=0"

let "nodoAunir=${#ips[@]}-1"

while [ $numvuelta -lt $vueltas ]
do
   echo "Vuelta $numvuelta"

let "i=0"
let "j=0"

./reiniciar.sh
echo "Esperando 65 seg. para que los routers esten listos"
sleep 65

for x in "${coordX[@]}"
do
   for x2 in "${coordX[@]}"
   do
      if [ $i -ne $j ]
      then
```

```
            c1=`echo "($x-$x2)^2" | bc`
            c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
            h=`echo "scale=0; sqrt($c1+$c2)" | bc`
            let "dist_max=${radios[$i]}+${radios[$j]}"
            if [ $h -lt $dist_max ]
            then
               R[$j]=1
            else
               R[$j]=0
            fi
         else
            R[$i]=0
         fi
         let "j++"
      done

      # Aqui ya tenemos R lleno para i
      let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
      ipv6R="3ffe:$numR::1"
      echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]} -"
      let "c=0"
      comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev wlan0;ip6tables
-P INPUT DROP\""
      eval "$comando"
      for valor in "${R[@]}"
      do
         if [ $valor -eq "1" ]
         then
            comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${macs[$c]} -j ACCEPT\""
            let "numC=($c+1)"
            echo "R$numC esta dentro de alcance"
            eval "$comando"
         fi
         let "c++"
      done

      # Ponemos a 0 el array
      let "v1=0"
      for v in "${R[@]}"
      do
         R[$v1]=0
         let "v1++"
      done

      let "j=0"
      let "i++"
done

let "i=0"
for x2 in "${coordX[@]}"
do
   if [ $nodoAunir != $i ]
   then
      comando="ssh root@${ips[$i]} \"olsrd\""
      eval "$comando" >>_log_tiempo_nodo_nuevo
   fi
```

```
   let "i++"
done

# ESPERAMOS A QUE SE ESTABILICEN LAS RUTAS
sleep 20

fin="false"
while [ $fin != "true" ]
do
   sleep 1
   let "i=0"

   # LANZO EL IP ROUTE PARA VER QUE TODOS LOS NODOS SON ACCESIBLES
   comando="ssh root@${ips[$i]} \"ip -6 route\""
   eval "$comando" > _tiempo_arranque_iproute
   rutas=(`cat _tiempo_arranque_iproute | grep '3ffe:' | awk '{ print $1 }'`)
   if [ ${#rutas[@]} == $nodoAunir ]
   then
      fin="true"
   fi
done

# INTRODUCIMOS EL NODO NUEVO Y COMPROBAMOS CUANDO SE ESTABILIZA LA
RED
comando="ssh root@${ips[$nodoAunir]} \"olsrd\""
eval "$comando" >>_log_tiempo_nodo_nuevo

tiempoInicio=`date +%s`

fin="false"
while [ $fin != "true" ]
do
   sleep 1
   let "i=0"

   # LANZO EL IP ROUTE PARA VER QUE TODOS LOS NODOS SON ACCESIBLES
   comando="ssh root@${ips[$i]} \"ip -6 route\""
   eval "$comando" > _tiempo_arranque_iproute
   rutas=(`cat _tiempo_arranque_iproute | grep '3ffe:' | awk '{ print $1 }'`)
   if [ ${#rutas[@]} == ${#ips[@]} ]
   then
      tiempoFin=`date +%s`
      let "tiempoTotal=$tiempoFin-$tiempoInicio"
      fin="true"
   fi
done

# Medimos el tiempo de inicio
# Tiramos la interfaz wifi del nodo
# Levantamos el nodo y lo configuramos como antes

echo "$tiempoTotal" >> _tiempos_nodo_nuevo
echo "El tiempo de arranque ha sido: $tiempoTotal"

let "numvuelta++"

done # fin del while
```

F.19 carga_sin_errores_Total.sh

```
#!/bin/bash

# Declaracion de variables

ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)
radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":" '{ print $2 }'`)

let "vueltas=3"
let "numvuelta=0"

./reiniciar.sh
echo "Esperando 65 seg. para que los routers esten listos"
sleep 65

sudo ip link set wlan0 down
sudo iwconfig wlan0 mode Ad-hoc channel 5
sudo ip link set wlan0 up
sudo ip ad add 3ffe:100::1/128 dev wlan0
echo "Estado de wlan0:"
echo "----------------"
iwconfig
echo "----------------"

let "i=0"
let "j=0"

for x in "${coordX[@]}"
do
   for x2 in "${coordX[@]}"
   do
      if [ $i -ne $j ]
      then
         c1=`echo "($x-$x2)^2" | bc`
         c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
         h=`echo "scale=0; sqrt($c1+$c2)" | bc`
         let "dist_max=${radios[$i]}+${radios[$j]}"
         if [ $h -lt $dist_max ]
         then
            R[$j]=1
         else
            R[$j]=0
         fi
      else
         R[$i]=0
      fi
      let "j++"
   done

   # Aqui ya tenemos R lleno para i
   let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
   ipv6R="3ffe:$numR::1"
```

```
   echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]} -"
   let "c=0"
   comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev wlan0;ip6tables
-P INPUT DROP\""
   eval "$comando"
   for valor in "${R[@]}"
   do
      if [ $valor -eq "1" ]
      then
         comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${macs[$c]} -j ACCEPT\""
         let "numC=($c+1)"
         echo "R$numC esta dentro de alcance"
         eval "$comando"
      fi
      let "c++"
   done
   comando="ssh root@${ips[$i]} \"olsrd\""
   eval "$comando"

   # Ponemos a 0 el array
   let "v1=0"
   for v in "${R[@]}"
   do
      R[$v1]=0
      let "v1++"
   done

   let "j=0"
   let "i++"
done

let "link=0"

sleep 20

for lnk in "${macs[@]}"
do

   while [ $numvuelta -lt $vueltas ]
   do
      echo "Capturando trafico Prueba $link de ${ips[$i]} Vuelta ($numvuelta de
$vueltas)"

      sudo tshark -i wlan0 -f "udp port 698" -a duration:60 -w /tmp/captura_tshark

      capinfos /tmp/captura_tshark >> _carga_sin_errores_Total
      echo "Datos almacenados en el fichero _carga_sin_errores_Total"

      sleep 5

      let "numvuelta++"

   done # fin del while

   echo "**************** FIN PRUEBA NODO ${macs[$link]}
********************" >> _carga_sin_errores_Total
```

```
    let "numvuelta=0"

    let "link++"

done
```

F.20 carga_sin_errores_por_nodo.sh

```
#!/bin/bash

# Declaracion de variables

ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)
radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":" '{ print $2 }'`)

let "numpruebas=0"
let "pruebas=1"

let "vueltas=20"
let "numvuelta=0"

./reiniciar.sh
echo "Esperando 65 seg. para que los routers esten listos"
sleep 65

sudo ip link set wlan0 down
sudo iwconfig wlan0 mode Ad-hoc channel 5
sudo ip link set wlan0 up
sudo ip ad add 3ffe:100::1/128 dev wlan0
echo "Estado de wlan0:"
echo "----------------"
iwconfig
echo "----------------"

let "i=0"
let "j=0"

for x in "${coordX[@]}"
do
    for x2 in "${coordX[@]}"
    do
        if [ $i -ne $j ]
        then
            c1=`echo "($x-$x2)^2" | bc`
            c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
            h=`echo "scale=0; sqrt($c1+$c2)" | bc`
            let "dist_max=${radios[$i]}+${radios[$j]}"
            if [ $h -lt $dist_max ]
            then
                R[$j]=1
            else
                R[$j]=0
            fi
        else
```

```
            R[$i]=0
        fi
        let "j++"
    done

    # Aqui ya tenemos R lleno para i
    let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
    ipv6R="3ffe:$numR::1"
    echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]} -"
    let "c=0"
    comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev wlan0;ip6tables
-P INPUT DROP\""
    eval "$comando"
    for valor in "${R[@]}"
    do
        if [ $valor -eq "1" ]
        then
            comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${macs[$c]} -j ACCEPT\""
            let "numC=($c+1)"
            echo "R$numC esta dentro de alcance"
            eval "$comando"
        fi
        let "c++"
    done
    comando="ssh root@${ips[$i]} \"olsrd\""
    eval "$comando"

    # Ponemos a 0 el array
    let "v1=0"
    for v in "${R[@]}"
    do
        R[$v1]=0
        let "v1++"
    done

    let "j=0"
    let "i++"
done

sleep 20

while [ $numpruebas -lt $pruebas ]
do

    let "link=0"

    for lnk in "${macs[@]}"
    do

        while [ $numvuelta -lt $vueltas ]
        do
            echo "Capturando trafico del nodo $link con MAC ${macs[$link]} Vuelta
($numvuelta de $vueltas)"

            sudo tshark -i wlan0 -f "udp port 698 and ether src ${macs[$link]}" -a
duration:60 -w /tmp/captura_tshark
```

```
            capinfos /tmp/captura_tshark >> _carga_sin_errores_por_Nodo
            echo "Datos almacenados en el fichero _carga_sin_errores_por_Nodo"

            sleep 5

            let "numvuelta++"

        done # fin del while

        echo "***************** FIN PRUEBA NODO ${macs[$link]}
********************" >> _carga_sin_errores_por_Nodo
        let "numvuelta=0"

        let "link++"

    done

    let "numpruebas++"

done
```

F.21 pico_carga_con_erroresv2.sh

```
#!/bin/bash

# Declaracion de variables

ips=(`cat nodos | grep 'x:' | awk '{ print $1 }'`)
macs=(`cat nodos | grep 'x:' | awk '{ print $2 }'`)
coordX=(`cat nodos | grep 'x:' | awk '{ print $3 }' | awk -F ":" '{ print $2 }'`)
coordY=(`cat nodos | grep 'y:' | awk '{ print $4 }' | awk -F ":" '{ print $2 }'`)
radios=(`cat nodos | grep 'r:' | awk '{ print $5 }' | awk -F ":" '{ print $2 }'`)

let "vueltas=20"
let "numvuelta=0"

#./reiniciar.sh
#echo "Esperando 65 seg. para que los routers esten listos"
#sleep 65

sudo ip link set wlan0 down
sudo iwconfig wlan0 mode Ad-hoc channel 5
sudo ip link set wlan0 up
sudo ip ad add 3ffe:100::1/128 dev wlan0
echo "Estado de wlan0:"
echo "----------------"
iwconfig
echo "----------------"

let "i=0"
let "j=0"

for x in "${coordX[@]}"
do
    for x2 in "${coordX[@]}"
    do
```

```
      if [ $i -ne $j ]
      then
         c1=`echo "($x-$x2)^2" | bc`
         c2=`echo "(${coordY[$i]}-${coordY[$j]})^2" | bc`
         h=`echo "scale=0; sqrt($c1+$c2)" | bc`
         let "dist_max=${radios[$i]}+${radios[$j]}"
         if [ $h -lt $dist_max ]
         then
            R[$j]=1
         else
            R[$j]=0
         fi
      else
         R[$i]=0
      fi
      let "j++"
   done

   # Aqui ya tenemos R lleno para i
   let "numR=($i+1)" # El numero del router es +1 porque empieza en 0
   ipv6R="3ffe:$numR::1"
   echo "- Procesando R$numR IP: ${ips[$i]} IPv6: $ipv6R MAC: ${macs[$i]} -"
   let "c=0"
   comando="ssh root@${ips[$i]} \"ip -6 addr add $ipv6R/128 dev wlan0;ip6tables
-P INPUT DROP\""
   eval "$comando"
   for valor in "${R[@]}"
   do
      if [ $valor -eq "1" ]
      then
         comando="ssh root@${ips[$i]} \"ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${macs[$c]} -j ACCEPT\""
         let "numC=($c+1)"
         echo "R$numC esta dentro de alcance"
         eval "$comando"
         # GUARDAMOS CADA UNO DE LOS ENLACES EN UN FICHERO PARA HACER
LUEGO LA PRUEBA
         if [ $i -lt $c ]
         then
            echo "${ips[$i]}-${macs[$c]}-${ips[$c]}-${macs[$i]}" >>
_pico_carga_enlaces
         fi
      fi
      let "c++"
   done
   comando="ssh root@${ips[$i]} \"olsrd\""
   eval "$comando"

   # Ponemos a 0 el array
   let "v1=0"
   for v in "${R[@]}"
   do
      R[$v1]=0
      let "v1++"
   done

   let "j=0"
```

```
   let "i++"
done

# AQUI YA LO TENEMOS LLENO PROCESAMOS EL FICHERO EN ARRAYS PARA IR
TIRANDO LOS ENLACES
# ENLACE A -> B
enlace_ips_N1=(`cat _pico_carga_enlaces | awk -F "-" '{ print $1 }'`)
enlace_mac_N2=(`cat _pico_carga_enlaces | awk -F "-" '{ print $2 }'`)
# ENLACE B -> A
enlace_ips_N2=(`cat _pico_carga_enlaces | awk -F "-" '{ print $3 }'`)
enlace_mac_N1=(`cat _pico_carga_enlaces | awk -F "-" '{ print $4 }'`)

let "link=0"

sleep 20

for lnk in "${enlace_ips_N1[@]}"
do

   while [ $numvuelta -lt $vueltas ]
   do
      echo "Enlace ($link de ${#enlace_ips_N1[@]}) Vuelta ($numvuelta de
$vueltas)"

      sudo tshark -i wlan0 -f "udp port 698" -a duration:60 -w /tmp/captura_tshark
&

      # AQUI TENEMOS QUE IR TIRANDO EL ENLACE 1 A 1

      # TIRAMOS EL ENLACE
      echo "MAC ${enlace_mac_N2[$link]} rechazada para
${enlace_ips_N1[$link]}"
      ssh root@${enlace_ips_N1[$link]} "ip6tables -D INPUT -i wlan0 -m mac --
mac-source ${enlace_mac_N2[$link]} -j ACCEPT"
      echo "MAC ${enlace_mac_N1[$link]} rechazada para
${enlace_ips_N2[$link]}"
      ssh root@${enlace_ips_N2[$link]} "ip6tables -D INPUT -i wlan0 -m mac --
mac-source ${enlace_mac_N1[$link]} -j ACCEPT"

      sleep 60

      capinfos /tmp/captura_tshark >> _pico_carga_con_errores_E
      echo "Datos almacenados en el fichero _pico_carga_con_errores_E"

      # LEVANTAMOS EL ENLACE
      echo "MAC ${enlace_mac_N2[$link]} aceptada para
${enlace_ips_N1[$link]}"
      ssh root@${enlace_ips_N1[$link]} "ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${enlace_mac_N2[$link]} -j ACCEPT"
      echo "MAC ${enlace_mac_N1[$link]} aceptada para
${enlace_ips_N2[$link]}"
      ssh root@${enlace_ips_N2[$link]} "ip6tables -A INPUT -i wlan0 -m mac --
mac-source ${enlace_mac_N1[$link]} -j ACCEPT"

      sleep 15

      let "numvuelta++"
```

```
    done # fin del while

    echo "FIN PRUEBA ENLACE ${enlace_ips_N1[$link]} - ${enlace_ips_N2[$link]}"
>> _pico_carga_con_errores_E
    let "numvuelta=0"

    let "link++"

done
```

APENDICE G – Presupuesto y Plan de Proyecto

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:
Antonio Guerrero Espartero
2.- Departamento:
Ingeniería Telemática
3.- Descripción del Proyecto:

- Titulo	**Estudio Experimental del funcionamiento de OSPF-MANET y OLSR en redes MMI**
- Duración (meses)	**12**
Tasa de costes Indirectos:	**20%**

4.- Presupuesto total del Proyecto (valores en Euros):
Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
					0,00	
					0,00	
Antonio Guerrero Espartero		Ingeniero	7,92	2.694,39	21.329,30	
					0,00	
					0,00	
		Hombres mes	**7,916190476**	**Total**	21.329,30	

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	%Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Ordenadores de sobremesa	2.200,00	100	12	60	440,00
Linksys WRT54GL	1.034,00	100	12	60	206,80
Cableado y alimentación	85,00	100	12	60	17,00
Material de oficina	60,00	100	12	12	60,00
		100		60	0,00
					0,00
				Total	723,80

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e]

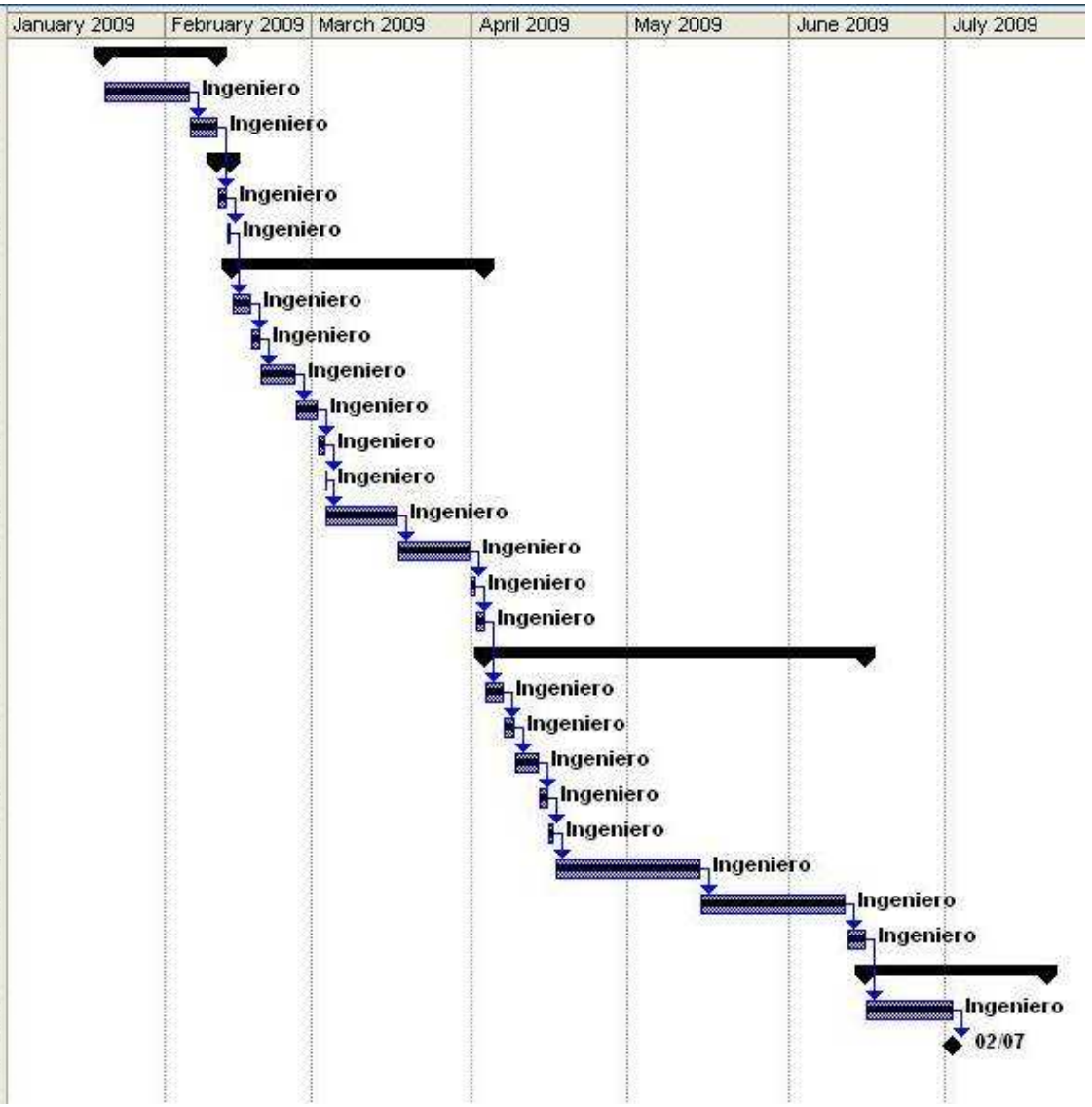
Descripción	Empresa	Costes imputable
	Total	0,00

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,…

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	21.329
Amortización	724
Subcontratación de tareas	0
Costes de funcionamiento	0
Costes Indirectos	4.411
Total	26.464

Task Name	Duration
Documentación y analisis del esta	**26 days?**
Estudio teorico general	20 days?
Estudio, busqueda de implementac	6 days?
Pruebas preeliminares	**3,6 days?**
Instalación de los PCs	2,4 days?
Pruebas de funcionamiento basico	1,2 days?
Primera prueba	**55,8 days?**
Estudio y analisis de los dispositiv	3 days?
Estudio del entorno Openwrt	3 days?
Diseño de la arquitectura software	8 days?
Compilación del firmware personal	3 days?
Preparar los primeros nueve route	2 days?
Diseño de la arquitectura de red pr	0,6 days?
Migración de los programas a los c	16 days?
Diseño de la bateria de pruebas	16 days?
Despliegue de la arquitectura de re	1 day?
Realización de la bateria de prueb	3,2 days?
Pruebas en escenarios definitivos	**82,4 days?**
Preparar los once routers restante	3 days?
Diseño de arquitectura final de pru	3 days?
Adaptación de las aplicaciones de	4 days?
Instalación y configuración de la pl	3 days?
Diseño de la bateria de pruebas	2,4 days?
Prueba definitiva OSPF-MANET	32 days?
Prueba definitva OLSR	32 days?
Analisis de los resultados	3 days?
Documentación y memoria del pro	**40 days?**
Documentación del desarrollo del p	20 days?
Memoria completa del proyecto	20 days?



Leganés, 12 de Julio de 2010

El ingeniero proyectista

BIBLIOGRAFIA Y REFERENCIAS

[TCPDUMP] *Sourceforce. Tcpdump Public Repository*. www.tcpdump.org [Consultado Mayo 2009]

[TSHARK] *tshark* www.wireshark.org/docs/man-pages/**tshark**.html [Consultado Mayo 2009]

[CAPINF] *capinfos* http://www.ethereal.com/docs/man-pages/capinfos.1.html [Consultado Mayo 2009]

[SCRIPTING] *Advanced Bash-Scripting Guide* http://www.tldp.org/LDP/abs/html/ [Consultado Abril 2009]

[WIFI] *IEEE 802.11 Wireless local area Networks*. http://www.ieee802.org/11 [Consultado Marzo 2009]

[AODV] *Ad hoc On-Demand Distance Vector (AODV) Routing RFC,* http://tools.ietf.org/html/rfc3561 [Consultado Mayo 2010]

[BATMAN] *Better approach to mobile ad-hoc networking*, http://tools.ietf.org/html/draft-wunderlich-openmesh-manet-routing-00 [Consultado Mayo 2010]

[DSR] *The Dynamic Source Routing Protocol (DSR). IETF RFC 4728*. http://tools.ietf.org/html/rfc4728 [Consultado Mayo 2010]

[FLOORNET] *FloorNet: A Wireless Multihop Testbed* http://floornet.org/ [Consultado Junio 2010]

[OLSR] T. Clausen, Ed., P. Jacquet, Ed. *Optimized Link State Routing Protocol (OLSR)*. *IETF RFC 3626*. Octubre 2003 http://www.ietf.org/rfc/rfc3626.txt [Consultado Abril 2010]

[OLSR2] INRIA – Unité de Recherche de Rocquencourt. OLSR http://hipercom.inria.fr/olsr/ [Consultado Febrero 2009]

[OLSRMPR] INRIA – Unité de Recherche de Rocquencourt. OLSR – *Demostration of MPR flooding*. http://hipercom.inria.fr/olsr/mpr-flooding.html [Consultado Febrero 2009]

[OLSRDIPV6] The Linux Documentation Project. *Linux Optimized Link State Routing Protocol (OLSR) Ipv6 HOWTO*. http://www.tldp.org/HOWTO/OLSR-IPv6-HOWTO/ [Consultado Abril 2009]

[OSPFv2] J. Moy *OSPF version 2, RFC 2328* http://www.ietf.org/rfc/rfc2328.txt [Consultado Febrero 2009]

[OSPFv3] R. Coltun *OSPF for IPv6* http://tools.ietf.org/html/rfc5340 [Consultado Febrero 2009]

[OSPFM] R. Ogier, P. Spagnolo. *Mobile Ad Hoc Network Extension of OSPF Using Connected Dominating Set (CDS) Flooding*. IETF RFC 5614. http://www.ietf.org/rfc/rfc5614.txt [Consultado Abril 2010]

[OSPFMDR] *R. Ogier. Advantages of OSPF-MDR.* 2006. http://home.earthlink.net/~ogier/mdr_position_draft_details.txt [Consultado Septiembre 2009]

[OSPFDBEO] R. Ogier. *OSPF Database Exchange Summary List Optimization* http://www.ietf.org/rfc/rfc5243.txt [Consultado Mayo 2009]

[WRT54] *Linksys WRT54GL OpenWRT Configuration* http://oldwiki.openwrt.org/OpenWrtDocs%282f%29Hardware%282f%29Linksys%282f%29WRT54GL.html [Consultado Enero 2009]

[OPENWRT] OpenWRT Firmware http://openwrt.org/ [Consultado Enero 2009]

[IPTABLES] IPTables http://www.netfilter.org/ [Consultado Febrero 2009]

[QUAGGA] Quagga http://www.quagga.net/ [Consultado Febero 2009]

[PARCHEMDR] *OSPF MANET MDR* http://hipserver.mct.phantomworks.org/ietf/ospf/ [Consultado Febrero 2009]

[OLSRD] Tonnesen, A. *OLSR.ORG* http://olsr.org/ [Consultado Feberero 2009]

[WMN07] Yan Zhang,Jijun Luo,Honglin Hu. *Wireless mesh networking: architectures, protocols and standards* 2007

[OLSRDPLG1] Tonnesen, A. *"Unik olsrd plugin implementation HOWTO"*. 2004 http://www.olsr.org/docs/olsrd-plugin-howto.html [Consultado Mayo 2010]

[OLSRDPLG2] Tonnesen, A.; Hafslund, A; Kure, A. *The Unik – OLSR plugin library*. OLSR Interop and Workshop, 2004. http://www.olsr.org/docs/olsr_plugin_paper.pdf [Consultado Mayo 2010]

[RIP] *Routing Information Protocol* http://rfc-ref.org/RFC-TEXTS/2453/index.html [Consultado Mayo 2010]

[RFC5613] A. Zinin *OSPF Link-Local Signaling* http://tools.ietf.org/html/rfc5613 [Consultado Mayo 2009]

[CDS] J. Blum, M. Ding *Connected Dominating Set in Sensor Networks and MANETs* pag. 334

[IANA] Internet Assigned Numbers Authority http://www.iana.org/ [Consultado Abril 2009]

[AEE] A. S. Tanenbaum *Computer Networks* 3 Ed. pag. 359-364

[DIJKS] W. Stallings *Data and Computer Communications* Ed. 7 pag. 385-387

[IPSEC] S. Kent *Security Architecture for the Internet Protocol* http://tools.ietf.org/html/rfc4301 [Consultado Mayo 2010]

[AVD] W. Stallings *Data and Computer Communications* Ed. 7 pag. 627-

628

[UBUNTU] Ubuntu http://releases.ubuntu.com/hardy/ [Consultado Febrero 2009]